Walmart Store Sales Forecasting

Walmart Recruiting - Store Sales Forecasting

The data comes from Kaggle, and there are altogether four data sets, respectively: **'train'**, **'test'**, **'feature'** and **'store'**.

https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/overview (https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/overview)

The purpose of this project is to analyze and model the data of each feature of three data sets including **'train', 'feature'** and **'store'**, and to predict the weekly sales volume from **2012-11-02** to **2013-08-01** with high accuracy and low error by using the established model and the data of each feature of 'Test' data set.

Import commonly used libraries

In [3]:
```
import numpy as np
import pandas as pd

#graphs - boxplot
import matplotlib as mpl
# Ignore warning messages
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import seaborn as sns
%matplotlib inline
from statsmodels.tsa.seasonal import seasonal_decompose
from pylab import rcParams

#graphs - missingno
import missingno as msno

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import mean_absolute_error
```

Import and observe data

Import Data

The parse_dates parameter converts the time character 'Date' string in a CSV file to Date format

In [4]:
```
# The parse_dates parameter converts the time character 'Date' string in a CSV file to Date format
train = pd.read_csv('/Users/zhiyongmai/Desktop/Walmart/walmart-recruiting-store-sales-forecasting/train.csv',
parse_dates=['Date'])
test = pd.read_csv('/Users/zhiyongmai/Desktop/Walmart/walmart-recruiting-store-sales-forecasting/test.csv',pa
rse_dates=['Date'])
store = pd.read_csv('/Users/zhiyongmai/Desktop/Walmart/walmart-recruiting-store-sales-forecasting/stores.csv'
)
feature = pd.read_csv('/Users/zhiyongmai/Desktop/Walmart/walmart-recruiting-store-sales-forecasting/features.
csv',parse_dates=['Date'])
```

Create a dictionary of dataset names

In [5]:
```
datasets = {'train':train,'test':test,'store':store,'features':feature}
```

Define a function and present the dataset

In [6]:
```
'''
Display tables side by side to save vertical space.
Input:
    dfs: list of pandas.DataFrame
    captions: list of table captions
'''
from IPython.core.display import display, HTML

def display_side_by_side(dfs:list, captions:list):
    output = ""
    combined = dict(zip(captions, dfs))
    for caption, df in combined.items():
        output += df.style.set_table_attributes("style='display:inline'").set_caption(caption)._repr_html_()
        output += "\xa0\xa0\xa0"
    display(HTML(output))
```

Display all dataset

In [7]:
```
display_side_by_side(
    [datasets[k].head() for k in datasets],
    [k for k in datasets])
```

train

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05 00:00:00	24924.500000	False
1	1	1	2010-02-12 00:00:00	46039.490000	True
2	1	1	2010-02-19 00:00:00	41595.550000	False
3	1	1	2010-02-26 00:00:00	19403.540000	False
4	1	1	2010-03-05 00:00:00	21827.900000	False

test

	Store	Dept	Date	IsHoliday
0	1	1	2012-11-02 00:00:00	False
1	1	1	2012-11-09 00:00:00	False
2	1	1	2012-11-16 00:00:00	False
3	1	1	2012-11-23 00:00:00	True
4	1	1	2012-11-30 00:00:00	False

store

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

features

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHolida
0	1	2010-02-05 00:00:00	42.310000	2.572000	nan	nan	nan	nan	nan	211.096358	8.106000	Fals
1	1	2010-02-12 00:00:00	38.510000	2.548000	nan	nan	nan	nan	nan	211.242170	8.106000	Tru
2	1	2010-02-19 00:00:00	39.930000	2.514000	nan	nan	nan	nan	nan	211.289143	8.106000	Fals
3	1	2010-02-26 00:00:00	46.630000	2.561000	nan	nan	nan	nan	nan	211.319643	8.106000	Fals
4	1	2010-03-05 00:00:00	46.500000	2.625000	nan	nan	nan	nan	nan	211.350143	8.106000	Fals

Understanding data

stores: This file contains anonymized information about the 45 stores, indicating the type and size of store.

train: This is the historical training data, which covers to 2010-02-05 to 2012-11-01. Within this file you will find the following features:

- Store - the store number
- Dept - the department number
- Date - the week
- Weekly_Sales - sales for the given department in the given store
- IsHoliday - whether the week is a special holiday week

test: This file is identical to train.csv, except it have been withheld the weekly sales. You must predict the sales for each triplet of store, department, and date in this file.

features: This file contains additional data related to the store, department, and activities for the given dates. It contains the following fearures:

- Store - the store number
- Date - the week
- Temperature - average temperature in the region
- Fuel_Price - cost of fuel in the region
- MarkDown1-5 - anonymized data related to promotional markdowns that Walmart is running. MarkDown data is only available after Nov 2011, and is not available for all stores all the time. Any missing value is marked with an NA.
- CPI - the consumer price index
- Unemployment - the unemployment rate
- IsHoliday - whether the week is a special holiday week

Data exploration and analysis

This part is mainly to explore the data, to help us understand and get familiar with the data, only really understand the data, we can better carry out the feature engineering and other processing, so this step is also very important.

1. Preliminary recognize of data:

Make preliminary exploration of data, check shapes, feature types, relevant statistics, etc. The preliminary data recognition part is the preliminary exploration of data. This part is mainly to have a general understanding of the imported data, including data simplification (head,tail, etc.), row and column information (shape), data statistical characteristics (describe), data information (info) and so on.

2. Perception data:

Further mining information on the basis of initial knowledge, mainly including data missing and abnormal conditions. The number of missing values and outliers can be printed, the main purpose of which is to determine whether the number of missing values and outliers is really large. If there are few, it is generally selected to fill, but if there are too many missing values and outliers, it can be considered to delete. Of course, what kind of processing to carry out, or according to the actual situation, otherwise it is easy to adversely affect the modeling.

Use box diagrams to determine outliers: Five-number generalization is commonly used in statistics to summarize data samples, which are: minimum, first quartile (lower quartile), second quartile (median), third quartile (upper quartile), and maximum. The first quartile Q1 is used as the bottom and the third quartile Q3 as the cover when drawing the box diagram. The middle boxes can be drawn (median is usually drawn as a marker). Interquartile spacing IQR = Q3-Q1, upper bound = Q3 + 1.5IQR, lower bound = Q1-1.5IQR. Values that exceed the upper and lower bounds are outliers.

3. Understand the data:

Further mining based on the first two, including looking at the distribution of predicted values and type judgment of characteristics. We usually look at the distribution of the data in terms of skewness and kurtosis, and we usually compare it to a normal distribution. We're going to treat the skewness and the kurtosis of the normal distribution as zero. If we calculate in practice that the kurtosis of skewness is not zero, that means that the variable is skewed to the left or skewed to the right, or that the variable has a high top and a flat top.

> * ***Skewness:*** A statistic that describes the distribution pattern of data. Skewness describes the symmetry of a population value distribution, which is simply the degree of data asymmetry. The skewness is calculated by the third order center distance:
>
> - Skewness = 0, the distribution form is the same as the Skewness of normal distribution.
> - Skewness > 0, with large positive deviation value, is Skewness or rightward deviation. The long tail is trailing to the right, and there are more extremes at the right end of the data.
> - Skewness < 0, with large negative deviation value, is negative deviation or left deviation. The long tail is trailing to the left, with more extreme values at the left end of the data.
> - The larger the absolute value is, the more asymmetric the data distribution is and the greater the degree of skew is.
>
> * ***Kurtosis:*** A statistic for describing the distribution pattern of all values of a variable. To put it simply, it is the degree of sharpness of the top of the data distribution.Kurtosis is calculated by the fourth order standard moment:
>
> - Kurtosis=0, which is the same as that of normal distribution.
> - Kurtosis> is more steep than a normal distribution of peak - the tip of the peak.
> - Kurtosis<0, than a normally distributed peak comes to the platform - a plateau.
>
> * ***Frequency distribution histogram:*** The height of the rectangle represents the ratio between the frequency of the corresponding group and the group distance (because the group distance is a constant, high frequency is usually used to represent the frequency directly for the convenience of drawing and looking at the graph). Such statistical graph is called frequency distribution histogram. It can: Clearly show the frequency distribution of each group; It is easy to show the difference of frequency between groups.

4. Preliminary mining data:

Mining numerical features and category features separately, including category skew, category distribution visualization, numerical correlation visualization, etc.

> - Heat Map: A matrix representation method in which the element values in the matrix are represented by colors and different colors represent values of different sizes. Color gives you an intuition of how big a value is at a particular location. You can also compare the color at that location to the color at any other location in the dataset. Thermal diagram is a very intuitive multivariate analysis method.
> - Correlation analysis: Check the direction and degree of the variation trend between two variables. The value range from -1 to +1 means that the two variables are not correlated, positive value means positive correlation, negative value means negative correlation, the greater the value, the stronger the correlation.

Analyze the store table

Information of the store table

In [8]:
```
store.head()
```

Out[8]:

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

In [9]:
```
store.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Store   45 non-null     int64 
 1   Type    45 non-null     object
 2   Size    45 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 1.2+ KB
```

Look at continuous variables using the describe( ) function

In [10]:
```
store.describe()
```

Out[10]:

	Store	Size
count	45.000000	45.000000
mean	23.000000	130287.600000
std	13.133926	63825.271991
min	1.000000	34875.000000
25%	12.000000	70713.000000
50%	23.000000	126512.000000
75%	34.000000	202307.000000
max	45.000000	219622.000000

Function describe( ) just display the numerical variable. But 'Type' is object. So let's do some other things to show it.

In [11]:
```
store.groupby('Type')['Store'].count()
```

Out[11]:
```
Type
A    22
B    17
C     6
Name: Store, dtype: int64
```

> Here we can see that there are more stores of type A and Type B.

We can also look at the details of 'Size' under 'Type'.

In [12]:
```
store.groupby('Type')['Size'].describe().T
```

Out[12]:

Type	A	B	C
count	22.000000	17.000000	6.000000
mean	177247.727273	101190.705882	40541.666667
std	49392.621098	32371.137916	1304.145033
min	39690.000000	34875.000000	39690.000000
25%	155840.750000	93188.000000	39745.000000
50%	202406.000000	114533.000000	39910.000000
75%	203819.000000	123737.000000	40774.000000
max	219622.000000	140167.000000	42988.000000

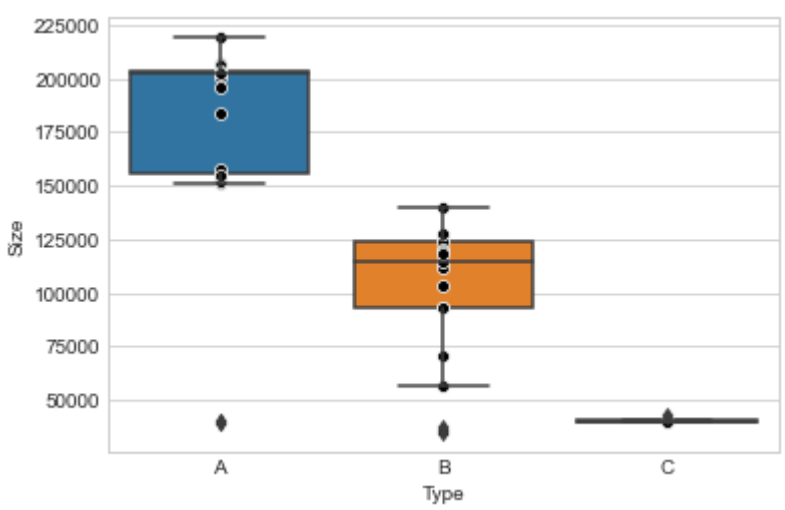
The following conclusions can be drawn from the above:

- The mean value of the store area frequency distribution of type 'A' stores is smaller than the median, which is a negative skewed distribution, also known as a left skewed distribution, with the peak shifting to the right and the long tail extending to the left.
- The mean value of shop area frequency distribution of 'B' and 'C' shops is not much different from the median value, that is, it tends to be approximately normal distribution.
- The std(standard deviation) of type 'A' and type 'B' is larger, covering a wide range. Based on their minimum, guess that there might be outliers.

Visualize the size of the store area grouped by type

In [13]:
```
sns.boxplot(x='Type',y='Size',data=store)
sns.scatterplot(x='Type',y='Size',data=store,color='black')
```

Out[13]: <AxesSubplot:xlabel='Type', ylabel='Size'>



We can see it clearly from this picture:

- Outliers of type 'A' and 'B' do exist
- Comparison of shop area of different types of stores: 'A' > 'B' > 'C'

We can see from the information presented in the Describe functions and from the box diagrams that type A and Type B are not only more numerous, but also larger in size.

Analyze the feature table

Information of feature table

In [14]:
```
feature.head(5)
```

Out[14]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
0	1	2010-02-05	42.31	2.572	NaN	NaN	NaN	NaN	NaN	211.096358	8.106	False
1	1	2010-02-12	38.51	2.548	NaN	NaN	NaN	NaN	NaN	211.242170	8.106	True
2	1	2010-02-19	39.93	2.514	NaN	NaN	NaN	NaN	NaN	211.289143	8.106	False
3	1	2010-02-26	46.63	2.561	NaN	NaN	NaN	NaN	NaN	211.319643	8.106	False
4	1	2010-03-05	46.50	2.625	NaN	NaN	NaN	NaN	NaN	211.350143	8.106	False

In [15]:
```
feature.tail(5)
```

Out[15]:

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHoliday
8185	45	2013-06-28	76.05	3.639	4842.29	975.03	3.00	2449.97	3169.69	NaN	NaN	False
8186	45	2013-07-05	77.50	3.614	9090.48	2268.58	582.74	5797.47	1514.93	NaN	NaN	False
8187	45	2013-07-12	79.37	3.614	3789.94	1827.31	85.72	744.84	2150.36	NaN	NaN	False
8188	45	2013-07-19	82.84	3.737	2961.49	1047.07	204.19	363.00	1059.46	NaN	NaN	False
8189	45	2013-07-26	76.06	3.804	212.02	851.73	2.06	10.88	1864.57	NaN	NaN	False

It can be seen that the data set named 'feature' has 12 features, among which the feature of 'Date' covers the Date of Friday of each week between 2010-02-05 and 2013-07-26.

Each feature in this table

In [16]:
```
feature.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype         
---  ------        --------------  -----         
 0   Store         8190 non-null   int64         
 1   Date          8190 non-null   datetime64[ns]
 2   Temperature   8190 non-null   float64       
 3   Fuel_Price    8190 non-null   float64       
 4   MarkDown1     4032 non-null   float64       
 5   MarkDown2     2921 non-null   float64       
 6   MarkDown3     3613 non-null   float64       
 7   MarkDown4     3464 non-null   float64       
 8   MarkDown5     4050 non-null   float64       
 9   CPI           7605 non-null   float64       
 10  Unemployment  7605 non-null   float64       
 11  IsHoliday     8190 non-null   bool          
dtypes: bool(1), datetime64[ns](1), float64(9), int64(1)
memory usage: 712.0 KB
```

This data set contains a total of 8,190 pieces of data, but there are only 3,000 to 4,000 pieces of data in each feature of the five promotion activities. There are a large number of missing values in these five fields, as well as missing values in the two features of unemployment rate and CPI.

Missing values in these features.

In [17]:
```
feature.isna().sum()
```

Out[17]:
```
Store              0
Date               0
Temperature        0
Fuel_Price         0
MarkDown1       4158
MarkDown2       5269
MarkDown3       4577
MarkDown4       4726
MarkDown5       4140
CPI              585
Unemployment     585
IsHoliday          0
dtype: int64
```

Numeric variables

Since individuals are more familiar with Celsius in the cognition of temperature, I will change the data of the 'Temperture' dimension represented by Fahrenheit to Celsius in order to observe and analyze this dimension.

In [18]:
```
feature['Temperature'] = (feature['Temperature']-32)*5/9
```

In [19]:
```
feature.describe().T
```

Out[19]:

	count	mean	std	min	25%	50%	75%	max
Store	8190.0	23.000000	12.987966	1.000000	12.000000	23.000000	34.000000	45.000000
Temperature	8190.0	15.197888	10.377004	-21.827778	7.723611	15.950000	23.266667	38.861111
Fuel_Price	8190.0	3.405992	0.431337	2.472000	3.041000	3.513000	3.743000	4.468000
MarkDown1	4032.0	7032.371786	9262.747448	-2781.450000	1577.532500	4743.580000	8923.310000	103184.980000
MarkDown2	2921.0	3384.176594	8793.583016	-265.760000	68.880000	364.570000	2153.350000	104519.540000
MarkDown3	3613.0	1760.100180	11276.462208	-179.260000	6.600000	36.260000	163.150000	149483.310000
MarkDown4	3464.0	3292.935886	6792.329861	0.220000	304.687500	1176.425000	3310.007500	67474.850000
MarkDown5	4050.0	4132.216422	13086.690278	-185.170000	1440.827500	2727.135000	4832.555000	771448.100000
CPI	7605.0	172.460809	39.738346	126.064000	132.364839	182.764003	213.932412	228.976456
Unemployment	7605.0	7.826821	1.877259	3.684000	6.634000	7.806000	8.567000	14.313000

It is clear that：

- The average of the five promotion activities is much larger than the median. It is possible that the data distribution of these features is skewed to the right. That is to say, the sales promotion will be very strong at a certain point. According to common sense. For example, there will be a very large sales promotion during Christmas or Thanksgiving to attract consumers on holiday. Or it is caused by a large number of missing values.
- Four of the five promotional activities have a negative minimum, which is irrational. If the promotion activity is negative, does that mean that consumers spend more on the same item because of the promotion? It's crazy.

Non-numeric variables

In [20]:
```
feature.groupby('IsHoliday')['Store'].count()/len(np.unique(feature['Store']))
```

Out[20]:
```
IsHoliday
False    169.0
True      13.0
Name: Store, dtype: float64
```

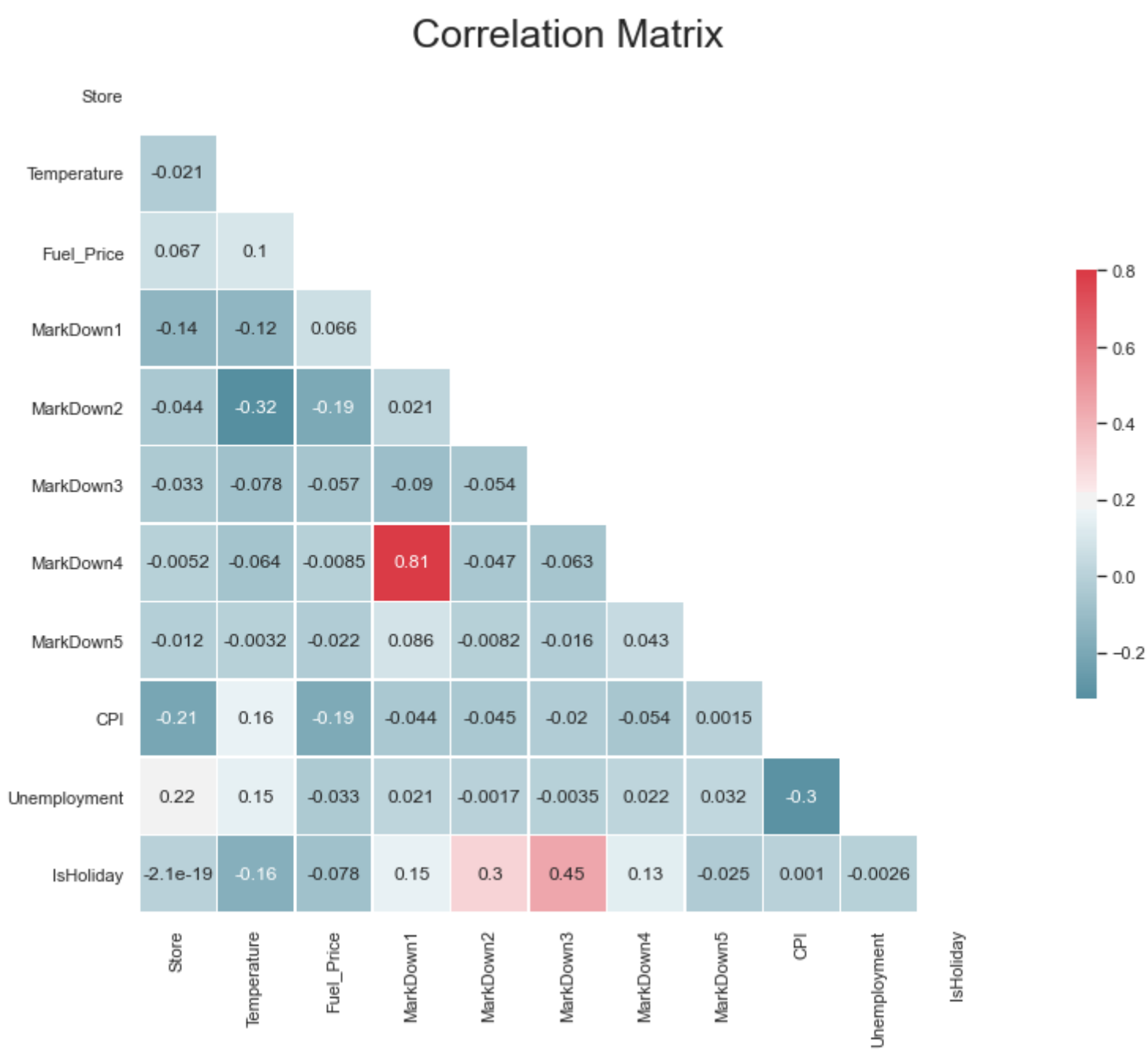
There are thirteen holiday points.

After the observation and analysis of the 'feature' table above, I have the following questions:

- Sales promotion is a marketing method aimed at attracting consumers, increasing sales volume. If such marketing method is not used in a specific place, specific time or appropriate opportunity, it will waste a lot of time and labor cost. So It's reasonable for me to wonder, is there a correlation between sales promotion and certain features? Examples include 'Store', 'Date', 'CPI', 'Unemployment' and 'IsHoliday'.
- Does CPI have anything to do with Unemployment? After all, from an economic point of view, when unemployment is high, the consumer price index tends to fall correspondingly.

Correlation of each feature in the feature

In [21]:
```
sns.set(style="white")

corr = feature.corr()

mask = np.triu(np.ones_like(corr, dtype=np.bool))

f, ax = plt.subplots(figsize=(18, 10))

cmap = sns.diverging_palette(220, 10, as_cmap=True)

plt.title('Correlation Matrix', fontsize=25)

sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.8, center=0.2,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
            annot=True)

plt.show()
```



It is clear that :

- Markdown1 and Markdown4 are highly correlated, so consider removing one.
- Except for Markdown5, there is a certain correlation between holiday and Markdowns, and Markdown3 has a high correlation with holiday.
- Only MarkDown1 has a strong negative correlation with Store, while other promotional activities have a weak correlation with Store.
- 'CPI' has a weak correlation with unemployment and promotional activity.
- The negative relationship between CPI and unemployment rate, conforms to economic logic.
- There is a negative correlation between CPI and store number. Can it be explained that the smaller (older) the store number is, the higher the regional consumption index (high consumption level) is?
- There was a higher negative correlation between temperature and MarkDown2.

Date - MarkDown

The relationship between the date and the promotion is unclear. Next, visually see if the date is related to these features.

In [22]:
```
def vis(var):
    plt.figure(figsize=(20,5))
    sns.lineplot(x="Date", y="{}".format(var), data=feature)
    plt.show()
```

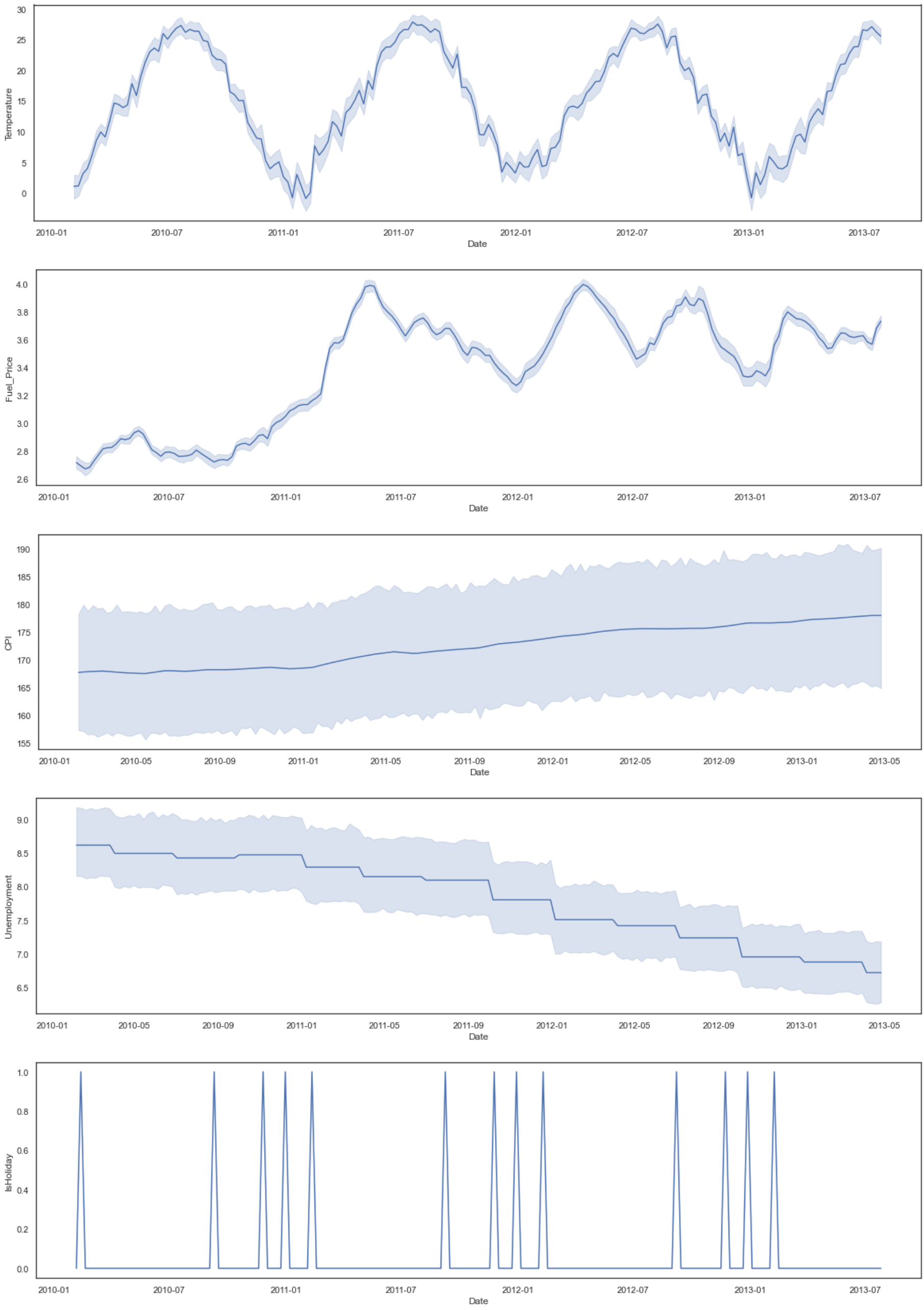
In [23]:
```
for var in feature.columns[4:9]:
    vis(var)
```

It is clear that :

- There were different periods of activity between Markdowns except for 1 and 4. In fact, the high correlation between MarkDown1 and MarkDown4 can also be inferred that the two promotions occurred at a similar time point.
- The special Markdown3 has only two peaks and both peaks are at the end of each year, so it is a special promotion during Thanksgiving and Christmas.
- Markdown1, 2, and 4 have a small peak in March and April, possibly for Easter. For the holiday variable, the analysis of Markdowns can be considered to include Easter, but the decision can only be made after the observation of weekly sales.
- Compared to other promotions, Markdown5 seems a bit lame. There is no particularly high peak, but there are many small peaks. It should be a daily promotion, which can be explained later by combining it with holiday and weekly sales.
- Markdowns are only available since November 2011, proving that there are many missing values.

Date - other features

In [24]:
```
for var in feature.columns[[2,3,9,10,11]]:
    vis(var)
```



It is clear that :

- We can see the obvious periodicity of temperature with date, which is consistent with our cognition.
- The price of fuel fluctuates greatly, but there is a clear upward trend.
- The residual of CPI is large, we can guess that different places (stores), CPI has a large difference, maybe have an impact on sales; At the same time, it also presents a relatively stable rising trend.
- The residual of Unemployment was large, which proved that it was very different in different places (stores) and might affect weekly sales. Showing a downward trend contrary to the CPI trend is in line with economic logic.

Analyze the train table

In [25]:
```
print('--------------train---------------')
train.info()
print('--------------store---------------')
store.info()
print('--------------feature---------------')
feature.info()
```

```
--------------train---------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Store         421570 non-null  int64
 1   Dept          421570 non-null  int64
 2   Date          421570 non-null  datetime64[ns]
 3   Weekly_Sales  421570 non-null  float64
 4   IsHoliday     421570 non-null  bool
dtypes: bool(1), datetime64[ns](1), float64(1), int64(2)
memory usage: 13.3 MB
--------------store---------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Store   45 non-null     int64
 1   Type    45 non-null     object
 2   Size    45 non-null     int64
dtypes: int64(2), object(1)
memory usage: 1.2+ KB
--------------feature---------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8190 entries, 0 to 8189
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         8190 non-null   int64
 1   Date          8190 non-null   datetime64[ns]
 2   Temperature   8190 non-null   float64
 3   Fuel_Price    8190 non-null   float64
 4   MarkDown1     4032 non-null   float64
 5   MarkDown2     2921 non-null   float64
 6   MarkDown3     3613 non-null   float64
 7   MarkDown4     3464 non-null   float64
 8   MarkDown5     4050 non-null   float64
 9   CPI           7605 non-null   float64
 10  Unemployment  7605 non-null   float64
 11  IsHoliday     8190 non-null   bool
dtypes: bool(1), datetime64[ns](1), float64(9), int64(1)
memory usage: 712.0 KB
```

In [26]:
```
train_new = train.merge(store,how='left').merge(feature,how='left')
```

To better present and observe the relationship between weekly sales dimensions and dates, add the following dimensions. Fields for year, month,week and quarter are generated based on Date.

In [27]:
```
train_new['Year'] = train_new['Date'].dt.year
train_new['Month'] = train_new['Date'].dt.month
train_new['Week'] = train_new['Date'].dt.week
train_new['Quarter'] = train_new['Date'].dt.to_period('Q')
result = train_new[['Date','Year','Month', 'Week','Quarter']].head()
print(result)
```

```
        Date  Year  Month  Week Quarter
0 2010-02-05  2010      2     5  2010Q1
1 2010-02-12  2010      2     6  2010Q1
2 2010-02-19  2010      2     7  2010Q1
3 2010-02-26  2010      2     8  2010Q1
4 2010-03-05  2010      3     9  2010Q1
```

In [28]:
```
train_new.head(3).append(train_new.tail(3))
```

Out[28]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4
0	1	1	2010-02-05	24924.50	False	A	151315	5.727778	2.572	NaN	NaN	NaN	NaN
1	1	1	2010-02-12	46039.49	True	A	151315	3.616667	2.548	NaN	NaN	NaN	NaN
2	1	1	2010-02-19	41595.55	False	A	151315	4.405556	2.514	NaN	NaN	NaN	NaN
421567	45	98	2012-10-12	1061.02	False	B	118221	12.483333	4.000	1956.28	NaN	7.89	599.32
421568	45	98	2012-10-19	760.01	False	B	118221	13.594444	3.969	2004.02	NaN	3.18	437.73
421569	45	98	2012-10-26	1076.80	False	B	118221	14.916667	3.882	4018.91	58.08	100.00	211.94

The data set of 'train_new' starts from the fifth week of 2010 and concludes in the 43rd week of 2012.

In [29]:
```
train_new.describe()
```

Out[29]:

	Store	Dept	Weekly_Sales	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3
count	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	150681.000000	111248.000000	137091.000000
mean	22.200546	44.260317	15981.258123	136727.915739	15.605588	3.361027	7246.420196	3334.628621	1439.421384
std	12.785297	30.492054	22711.183519	60980.583328	10.248851	0.458515	8291.221345	9475.357325	9623.078290
min	1.000000	1.000000	-4988.940000	34875.000000	-18.922222	2.472000	0.270000	-265.760000	-29.100000
25%	11.000000	18.000000	2079.650000	93638.000000	8.155556	2.933000	2240.270000	41.600000	5.080000
50%	22.000000	37.000000	7612.030000	140167.000000	16.716667	3.452000	5347.450000	192.000000	24.600000
75%	33.000000	74.000000	20205.852500	202505.000000	23.488889	3.738000	9210.900000	1926.940000	103.990000
max	45.000000	99.000000	693099.360000	219622.000000	37.855556	4.468000	88646.760000	104519.540000	141630.610000

There are negative values in 'Weekly_Sales', but the sales can't be negative, so these are outliers.

Information of the 'train_new' dataset

In [30]:
```
train_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 421570 entries, 0 to 421569
Data columns (total 20 columns):
 #   Column         Non-Null Count   Dtype         
---  ------         --------------   -----         
 0   Store          421570 non-null  int64         
 1   Dept           421570 non-null  int64         
 2   Date           421570 non-null  datetime64[ns]
 3   Weekly_Sales   421570 non-null  float64       
 4   IsHoliday      421570 non-null  bool          
 5   Type           421570 non-null  object        
 6   Size           421570 non-null  int64         
 7   Temperature    421570 non-null  float64       
 8   Fuel_Price     421570 non-null  float64       
 9   MarkDown1      150681 non-null  float64       
 10  MarkDown2      111248 non-null  float64       
 11  MarkDown3      137091 non-null  float64       
 12  MarkDown4      134967 non-null  float64       
 13  MarkDown5      151432 non-null  float64       
 14  CPI            421570 non-null  float64       
 15  Unemployment   421570 non-null  float64       
 16  Year           421570 non-null  int64         
 17  Month          421570 non-null  int64         
 18  Week           421570 non-null  int64         
 19  Quarter        421570 non-null  period[Q-DEC] 
dtypes: bool(1), datetime64[ns](1), float64(10), int64(6), object(1), period[Q-DEC](1)
memory usage: 64.7+ MB
```

The missing value in the promotion activities has not been dealt with yet, so let's leave it alone and deal with it later. Remember that when the feature data set was analyzed separately in the previous part, there were missing values in the two features of 'CPI' and 'Unemployment'. In this case, the missing values actually disappeared, and it was speculated that these missing values only existed after 2012-11-02 (test dataset).

Distribution of weekly sales

Outliers in 'weekly_Sales' first

In [31]:
```
sales_medians = train_new['Weekly_Sales'].median()
train_new['Weekly_Sales'] = train_new['Weekly_Sales'].apply(lambda x:np.where(x<0,sales_medians,x))
train_new['Weekly_Sales'] = train_new['Weekly_Sales'].apply(lambda x:np.where(x==0,1,x))
train_new.describe()
```

Out[31]:

	Store	Dept	Weekly_Sales	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3
count	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	150681.000000	111248.000000	137091.000000
mean	22.200546	44.260317	16004.669877	136727.915739	15.605588	3.361027	7246.420196	3334.628621	1439.421384
std	12.785297	30.492054	22698.578122	60980.583328	10.248851	0.458515	8291.221345	9475.357325	9623.078290
min	1.000000	1.000000	0.010000	34875.000000	-18.922222	2.472000	0.270000	-265.760000	-29.100000
25%	11.000000	18.000000	2130.877500	93638.000000	8.155556	2.933000	2240.270000	41.600000	5.080000
50%	22.000000	37.000000	7612.030000	140167.000000	16.716667	3.452000	5347.450000	192.000000	24.600000
75%	33.000000	74.000000	20205.852500	202505.000000	23.488889	3.738000	9210.900000	1926.940000	103.990000
max	45.000000	99.000000	693099.360000	219622.000000	37.855556	4.468000	88646.760000	104519.540000	141630.610000

The outliers have been replaced, and there are no negative values in 'Weekly_Sales'. And in order to try the log transformation, I'm going to replace all the zeros with the number one, because when x is equal to zero, the log transformation is negative infinity.

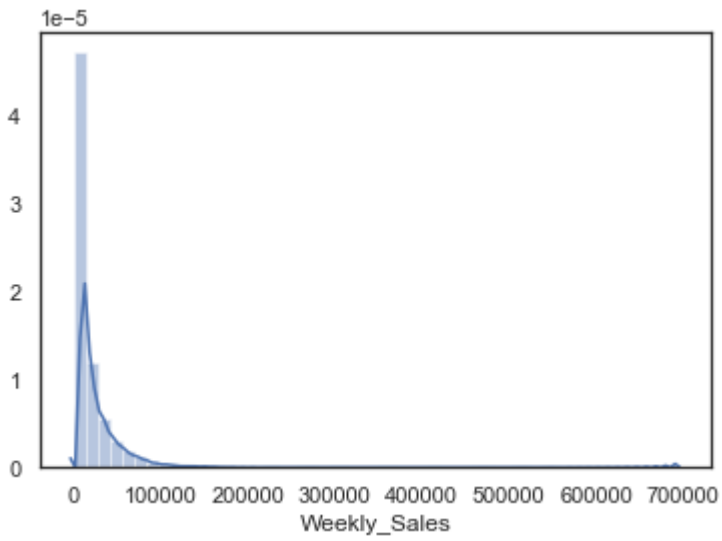
Frequency distribution

Check for:

1. frequency distribution
2. kernel density estimation
3. skewness
4. kurtosis

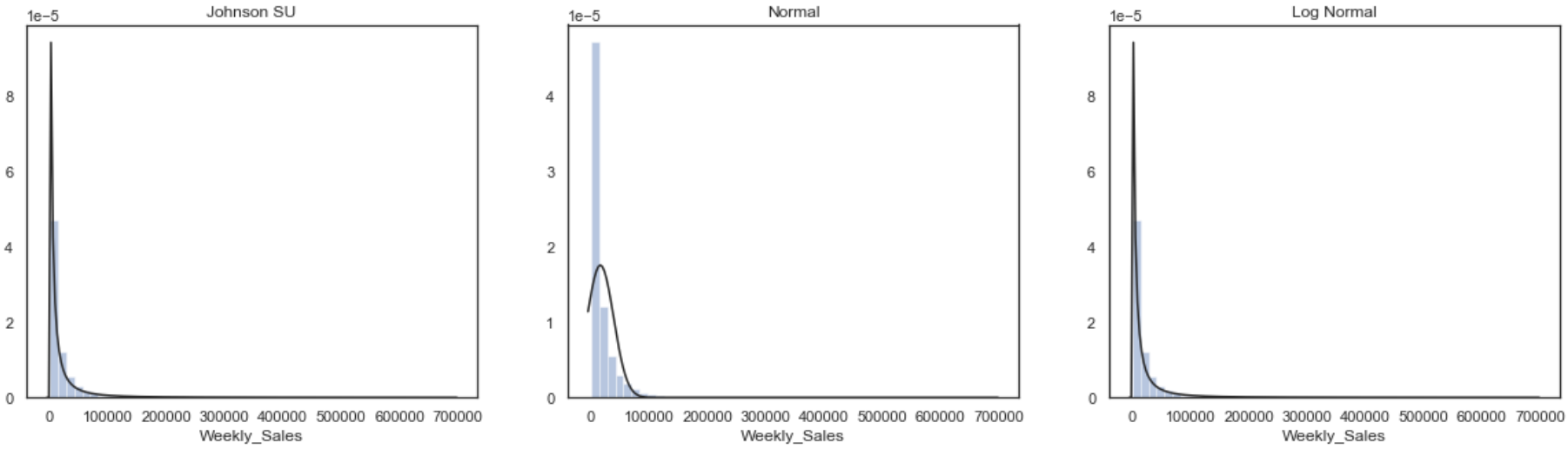
In [32]:
```
sns.distplot(train_new['Weekly_Sales'])
print('Skewness: %f' % train_new['Weekly_Sales'].skew())
print('Kurtosis: %f' % train_new['Weekly_Sales'].kurt())
```

```
Skewness: 3.265277
Kurtosis: 21.531554
```



In [33]:
```
import scipy.stats as st
y = train_new['Weekly_Sales']

plt.figure(figsize=(20,5))
plt.subplot(131);
plt.title('Johnson SU')
sns.distplot(y, kde=False, fit=st.johnsonsu)

plt.subplot(132);
plt.title('Normal')
sns.distplot(y, kde=False, fit=st.norm)

plt.subplot(133);
plt.title('Log Normal')
sns.distplot(y, kde=False, fit=st.lognorm)
```

Out[33]: <AxesSubplot:title={'center':'Log Normal'}, xlabel='Weekly_Sales'>

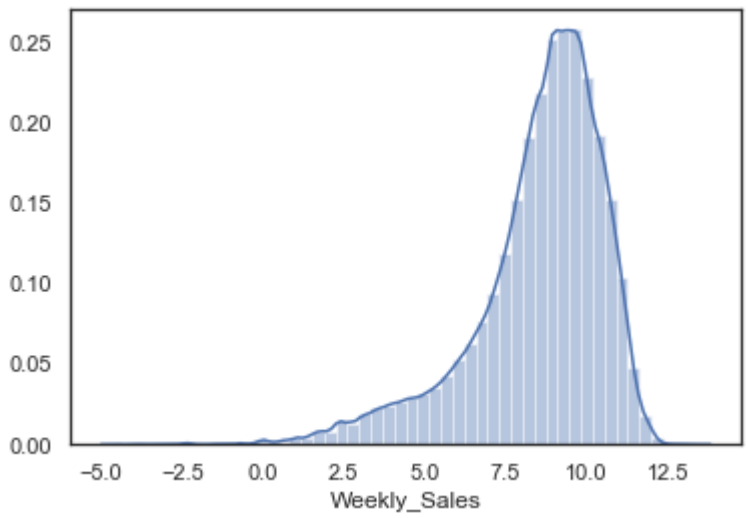


> The log transformation of the prediction label can make it more obedient to the normal distribution, which is conducive to modeling.

Try log conversion

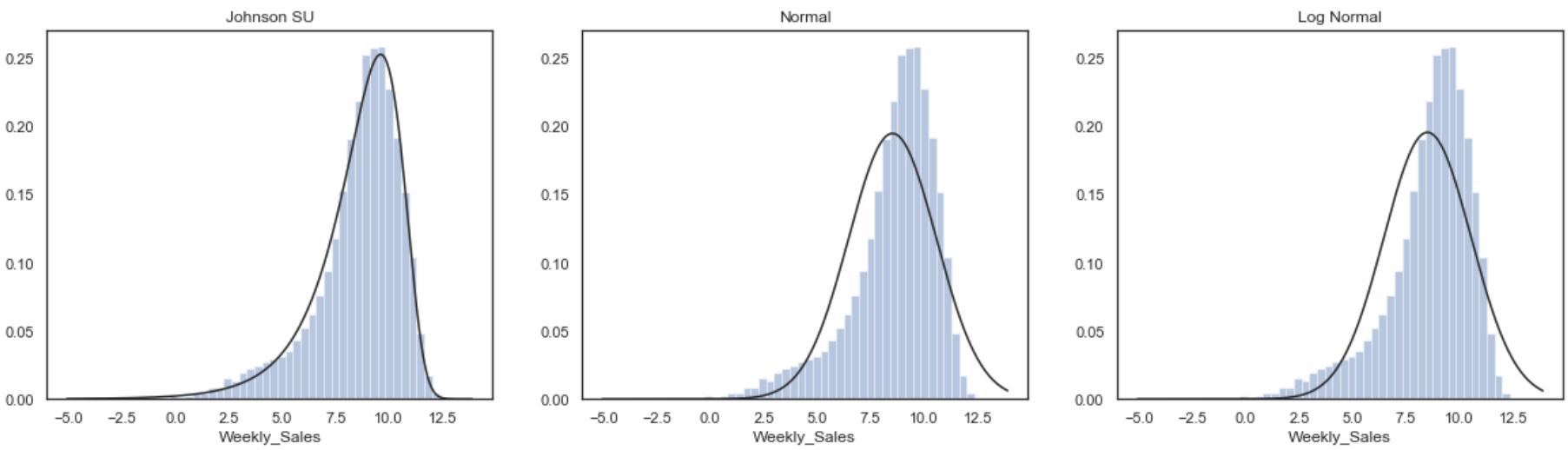
In [34]:
```
sns.distplot(np.log(train_new['Weekly_Sales']))
print('Skewness: %f' % np.log(train_new['Weekly_Sales']).skew())
print('Kurtosis: %f' % np.log(train_new['Weekly_Sales']).kurt())
```

```
Skewness: -1.313834
Kurtosis: 2.256701
```



In [35]:
```
y = np.log(train_new['Weekly_Sales'])

plt.figure(figsize=(20,5))
plt.subplot(131);
plt.title('Johnson SU')
sns.distplot(y, kde=False, fit=st.johnsonsu)

plt.subplot(132);
plt.title('Normal')
sns.distplot(y, kde=False, fit=st.norm)

plt.subplot(133);
plt.title('Log Normal')
sns.distplot(y, kde=False, fit=st.lognorm)
```

Out[35]: <AxesSubplot:title={'center':'Log Normal'}, xlabel='Weekly_Sales'>

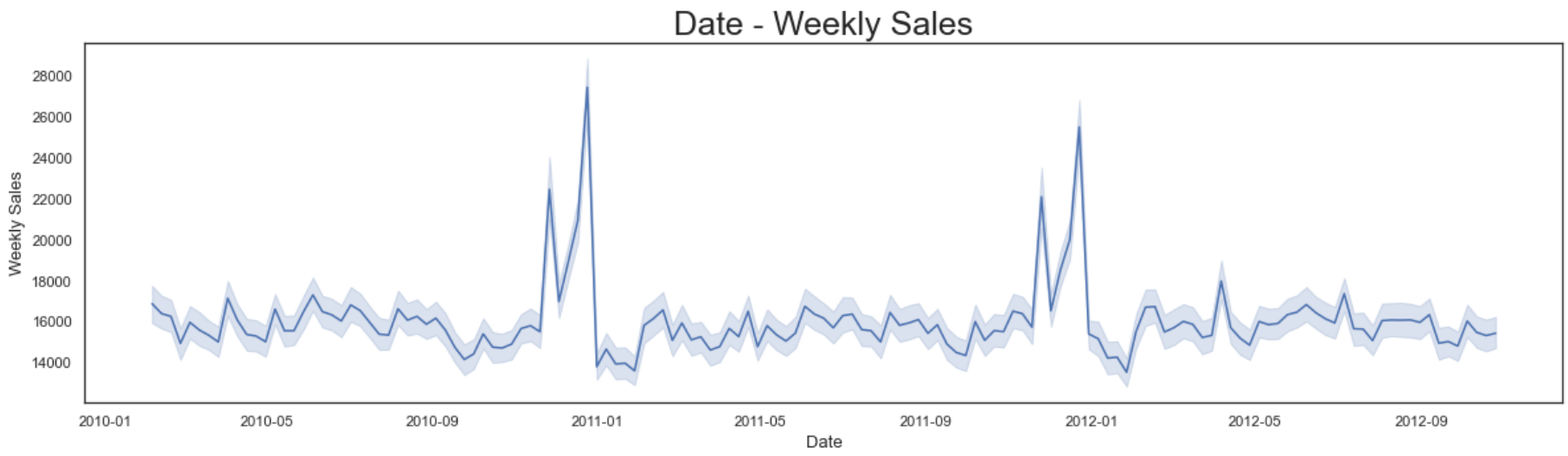


The best fit is the unbounded Johnson distribution.

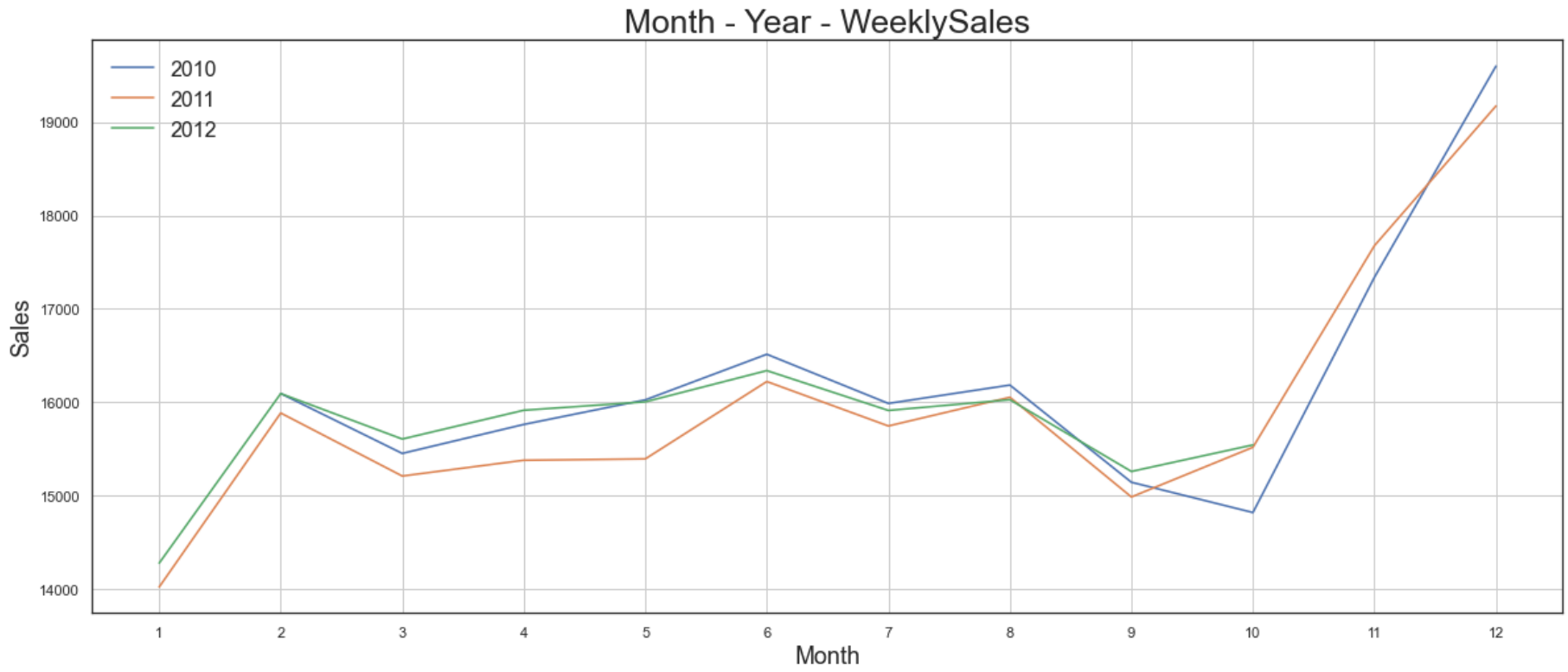
Date - Weekly sales

The line chart

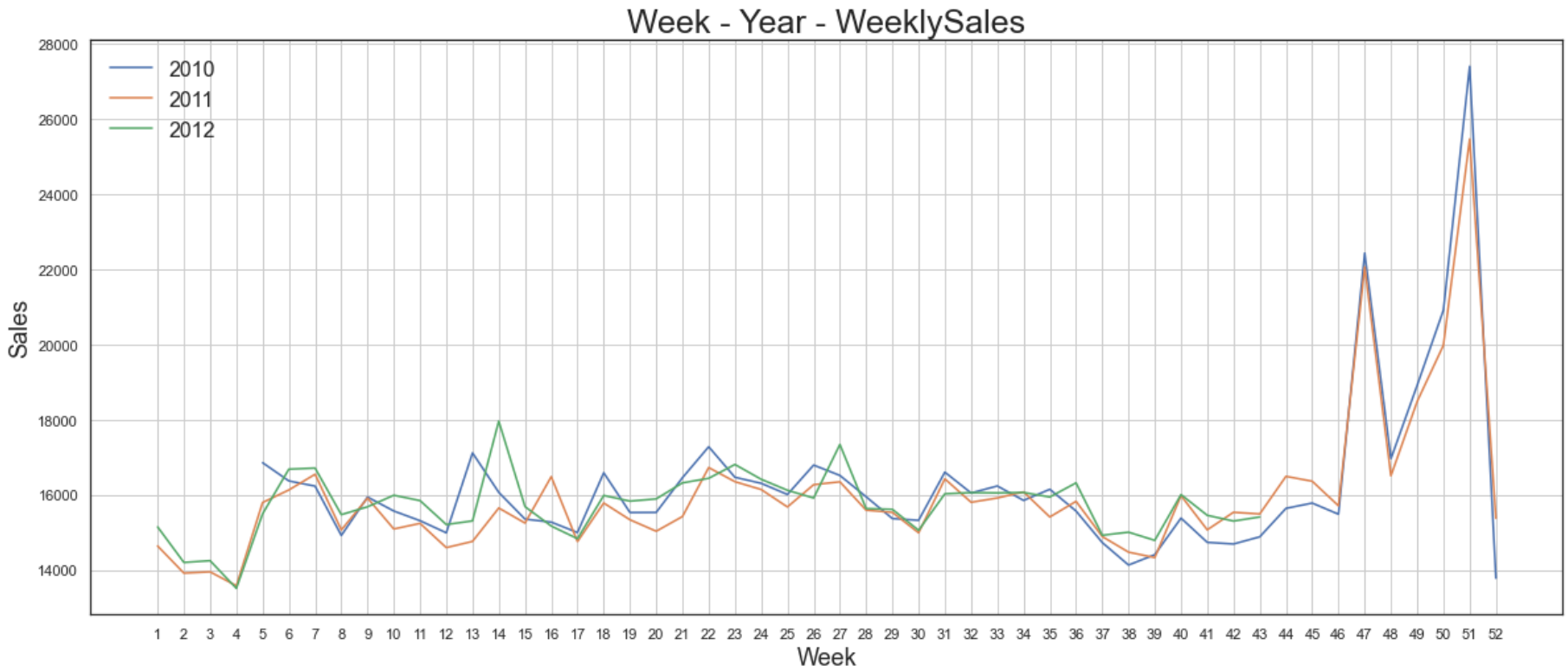
In [36]:
```
plt.figure(figsize=(20,5))
sns.lineplot(x="Date", y="Weekly_Sales", data=train_new)
plt.title('Date - Weekly Sales',fontsize=25)
plt.xlabel('Date',fontsize=13)
plt.ylabel('Weekly Sales',fontsize=13)
plt.show()
```



In [37]:
```
weekly_sales_2010 = train_new[train_new['Year']==2010]["Weekly_Sales"].groupby(train_new['Month']).mean()
weekly_sales_2011 = train_new[train_new['Year']==2011]["Weekly_Sales"].groupby(train_new['Month']).mean()
weekly_sales_2012 = train_new[train_new['Year']==2012]["Weekly_Sales"].groupby(train_new['Month']).mean()
plt.figure(figsize=(20,8))
sns.lineplot(weekly_sales_2010.index,weekly_sales_2010.values)
sns.lineplot(weekly_sales_2011.index,weekly_sales_2011.values)
sns.lineplot(weekly_sales_2012.index,weekly_sales_2012.values)
plt.legend(['2010', '2011', '2012'], loc='best', fontsize=16)
plt.title('Month - Year - WeeklySales',fontsize=25)
plt.xticks(np.arange(1,13,step=1))
plt.ylabel('Sales', fontsize=18)
plt.xlabel('Month', fontsize=18)
plt.grid()
plt.show()
```



In [38]:
```
weekly_sales_2010 = train_new[train_new['Year']==2010]["Weekly_Sales"].groupby(train_new['Week']).mean()
weekly_sales_2011 = train_new[train_new['Year']==2011]["Weekly_Sales"].groupby(train_new['Week']).mean()
weekly_sales_2012 = train_new[train_new['Year']==2012]["Weekly_Sales"].groupby(train_new['Week']).mean()
plt.figure(figsize=(20,8))
sns.lineplot(weekly_sales_2010.index,weekly_sales_2010.values)
sns.lineplot(weekly_sales_2011.index,weekly_sales_2011.values)
sns.lineplot(weekly_sales_2012.index,weekly_sales_2012.values)
plt.legend(['2010', '2011', '2012'], loc='best', fontsize=16)
plt.title('Week - Year - WeeklySales',fontsize=25)
plt.xticks(np.arange(1,53,step=1))
plt.ylabel('Sales', fontsize=18)
plt.xlabel('Week', fontsize=18)
plt.grid()
plt.show()
```



According to the above three line charts, weekly sales change with the annual development, weekly sales change with the monthly development and weekly sales change with the annual development, and the following information can be obtained:

- Except for November and December, weekly sales are relatively stable in most of the year. Although there are some fluctuations in weekly sales, the fluctuation is not large.
- There was a big drop in sales in the week after Christmas in December, with January being the lowest of the year.
- In the line chart of the change of weekly sales with the development of the week, it can be found that there is usually an upward trend of about $2,000 in weekly sales in the 13th and 14th weeks. However, compared with the fluctuation in November and December, such a small fluctuation is not worthy of attention. Therefore, the effect of Easter on promoting consumption is not significant and does not need to be added to the holiday dimension.
- September and October (from the 37th to the 43rd week) were also relatively low weekly sales. Maybe everyone is saving up for the November and December sales.

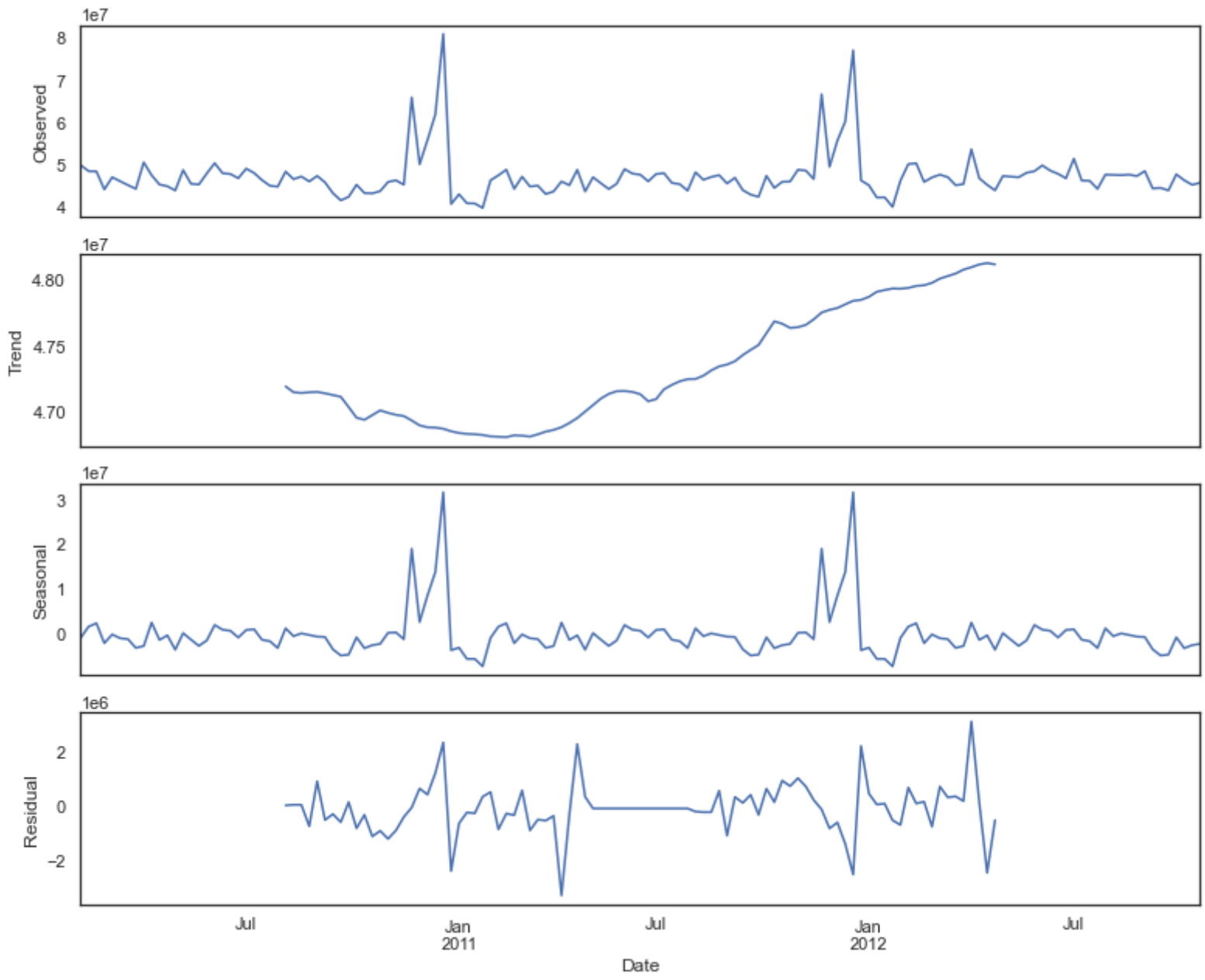
Seasonal trends

In this approach, trends and seasonality are modeled separately and the rest of the sequence is returned.

- Freq =52 indicates a period of 52, because there are 52 weeks in a year.
- Seasonal decomposition using moving averages.

The moving average method is a simple and smooth forecasting technique. Its basic idea is to calculate the chronological average of a certain number of items in order to reflect the long-term trend according to the data of time series. Therefore, when a change in value due to the time sequence cycle and the influence of random fluctuations, ups and downs is bigger, not easy to show the development trend of events, using moving average method can eliminate the influence of these factors and show the events development direction and trend (that is, the trend line), and then according to the analysis and prediction of the trend line sequence of long-term trend.

In [39]:
```
train_ts = train_new.groupby('Date').sum()["Weekly_Sales"]
rcParams['figure.figsize'] = 11, 9
seasonal = seasonal_decompose(train_ts,freq=52)
seasonal.plot()
plt.show()
```



As can be seen:

- In the graph of periodicity, weekly sales of each year are also significantly cyclical only during the two holidays of Thanksgiving and Christmas.
- With the development of the weekly sales, there is a clear upward trend.
- With the exception of Thanksgiving in November and Christmas in December, the "Easter" holiday is really not that important for weekly sales, although the residuals fluctuate considerably in April. We could even consider deleting other holidays in the data cleansing section.

Features

In [40]:
```
train_new.describe()
```

Out[40]:

	Store	Dept	Weekly_Sales	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3
count	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	421570.000000	150681.000000	111248.000000	137091.000000
mean	22.200546	44.260317	16004.669877	136727.915739	15.605588	3.361027	7246.420196	3334.628621	1439.421384
std	12.785297	30.492054	22698.578122	60980.583328	10.248851	0.458515	8291.221345	9475.357325	9623.078290
min	1.000000	1.000000	0.010000	34875.000000	-18.922222	2.472000	0.270000	-265.760000	-29.100000
25%	11.000000	18.000000	2130.877500	93638.000000	8.155556	2.933000	2240.270000	41.600000	5.080000
50%	22.000000	37.000000	7612.030000	140167.000000	16.716667	3.452000	5347.450000	192.000000	24.600000
75%	33.000000	74.000000	20205.852500	202505.000000	23.488889	3.738000	9210.900000	1926.940000	103.990000
max	45.000000	99.000000	693099.360000	219622.000000	37.855556	4.468000	88646.760000	104519.540000	141630.610000

We can see:

- In the feature of departments, the mean value is greater than the median. It is speculated that not every walmart store has the same number and type of departments. Also, departments are numbered from 1 to 99, and it is not known whether there is a fault in the middle.
- The mean of the weekly sales is much larger than the median, indicating that a small number of stores have relatively high weekly sales compared to most other stores. At the same time, the minimum value of the word dimension is actually a negative value, so we can know that there should be outliers in this field.
- For the temperature, the maximum is 37.9 ° C and the minimum is -18.9 ° C. In fact, when the temperature is above 30 degrees Celsius and below zero degrees Celsius, such as the extreme weather, people are usually less willing to go out. But driving a car is out of the question. Therefore, whether the temperature has any influence on the weekly sales volume needs to be carefully observed.
- For the fuel price, its standard deviation is not large, indicating that the fluctuation of fuel price is not very large during the period of 2010-2012. Although an upward trend can be seen from the visualized graph of weekly sales, this rising price is actually not very much and does not necessarily affect People's Daily consumption habits.
- MarkDown1-5,'CPI','Employment',they are pretty much the same as the previous analysis.

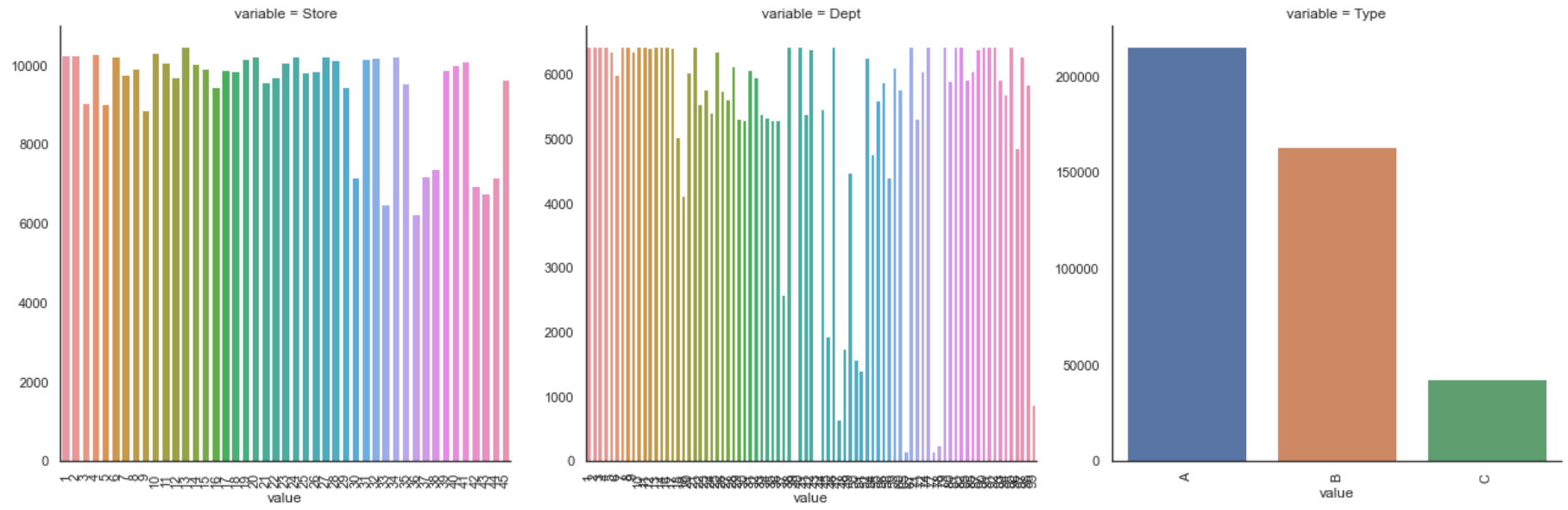
Categorical features and Continuous features

In [41]:
```
continuous_features = ['Size','Temperature','Fuel_Price','CPI','Unemployment']
continuous_features.extend(['MarkDown'+str(i) for i in range(1,6)])
categorical_features = ['Store','Dept','Type',]
print('continuous_features:')
print(continuous_features)
print('categorical_features:')
print(categorical_features)
```

```
continuous_features:
['Size', 'Temperature', 'Fuel_Price', 'CPI', 'Unemployment', 'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDow
n4', 'MarkDown5']
categorical_features:
['Store', 'Dept', 'Type']
```

Categorical features

In [42]:
```
def count_plot(x,  **kwargs):
    sns.countplot(x=x)
    x=plt.xticks(rotation=90)

f = pd.melt(train_new,value_vars=categorical_features)
g = sns.FacetGrid(f,col="variable",col_wrap=3,sharex=False,sharey=False,size=6,palette='dark')
g = g.map(count_plot, "value")
```



It can be found that:

- There are certain departments in certain stores.
- There are more 'A' and 'B' types of shops.

Continuous features

Distribution of each continuous feature

In [43]:
```
continuous_train_new = train_new[continuous_features]
```

Look at the skewness and kurtosis of several continuous features.

In [44]:
```
for col in continuous_features:
    print('{:15}'.format(col),
          'Skewness: {:05.2f}'.format(continuous_train_new[col].skew()) ,
          '    ' ,
          'Kurtosis: {:06.2f}'.format(continuous_train_new[col].kurt())
         )
```

```
Size            Skewness: -0.33     Kurtosis: -01.21
Temperature     Skewness: -0.32     Kurtosis: -00.64
Fuel_Price      Skewness: -0.10     Kurtosis: -01.19
CPI             Skewness: 00.09     Kurtosis: -01.83
Unemployment    Skewness: 01.18     Kurtosis: 002.73
MarkDown1       Skewness: 03.34     Kurtosis: 017.61
MarkDown2       Skewness: 05.44     Kurtosis: 037.59
MarkDown3       Skewness: 08.40     Kurtosis: 077.69
MarkDown4       Skewness: 04.85     Kurtosis: 030.00
MarkDown5       Skewness: 08.17     Kurtosis: 107.85
```

The markdown1-5 has greater skewness and kurtosis, especially the MarkDown3 and MarkDown5.

pd.melt:

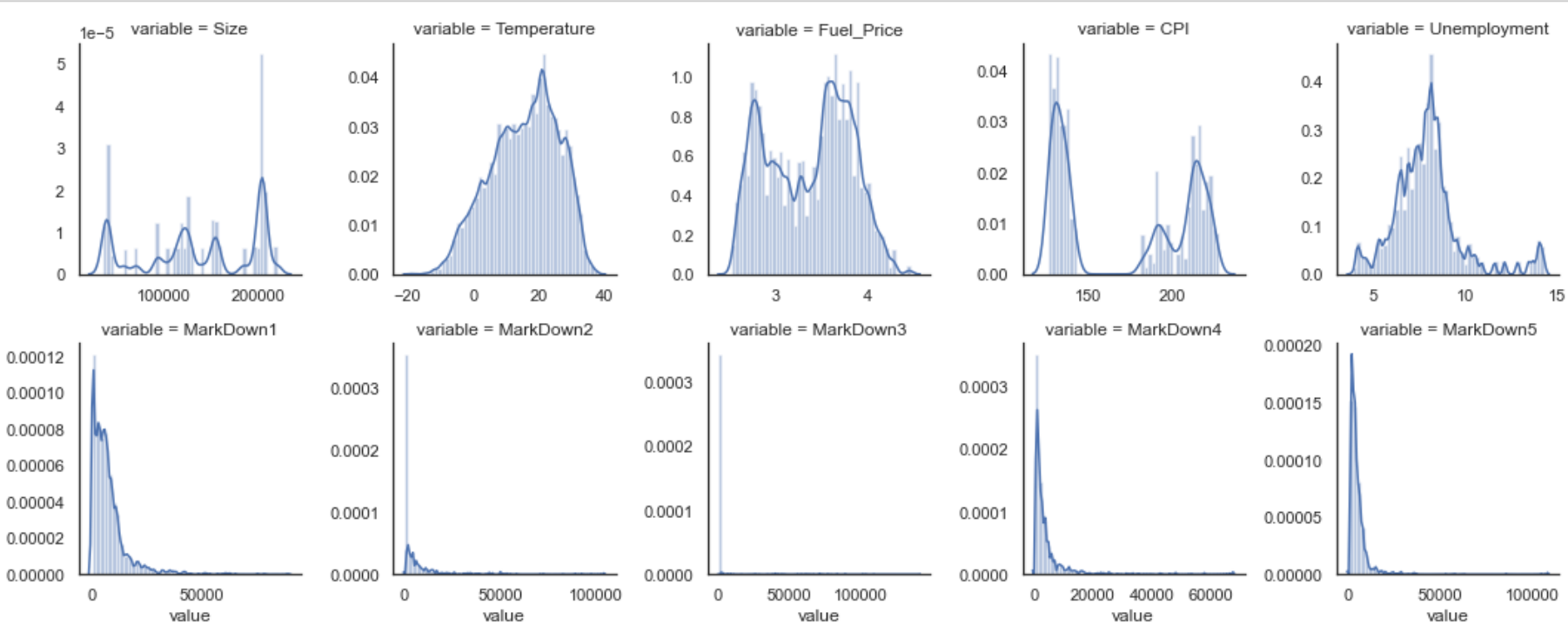
1. df.melt( ) is the df.pivot( ) reversal operation function that that converts the column name to the column values(columns name → column values) to refactor the DataFrame.
2. If the df.pivot( ) converts the long data set to the wide data set, the df.melt( ) converts the wide data set to the long data set.
3. id_vars: Identity column.
4. value_vars: The columns that need to be converted, if not specified, are converted except for id_vars.

sns.distplot:

1. kde: Kernel density estimation is a non-parametric estimation method, which averages out a known density function, namely kernel, at the observation point in order to obtain a smooth estimation curve.
2. hist: histogram
3. The API can draw histogram and kernel density estimation maps respectively, or a composite map of histogram and kernel density estimation maps. By default, the composite map is drawn, and the Settings are as follows:

   - hist =True: means to draw the histogram (default is True), or not if it is False.
   - kde =True: means to draw the kernel density estimation graph (default is True), or if False, then draw.

In [45]:
```
f = pd.melt(train_new, value_vars=continuous_features)
g = sns.FacetGrid(f, col="variable",  col_wrap=5, sharex=False, sharey=False)
g = g.map(sns.distplot, "value", kde=True, hist=True)
```



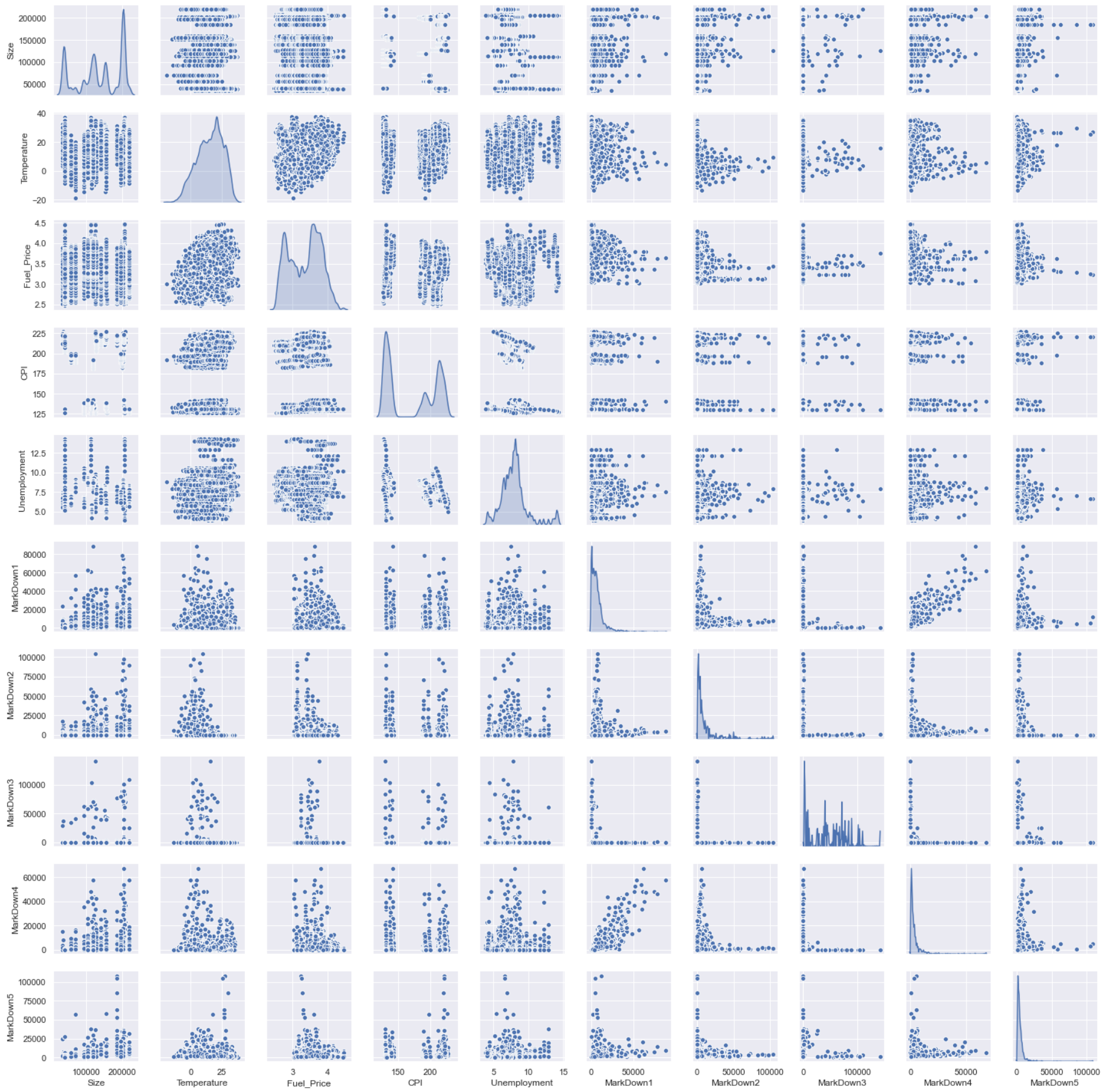
Only temperature values tend to be normally distributed.

The relationship between continuous features

sns.pairplot:

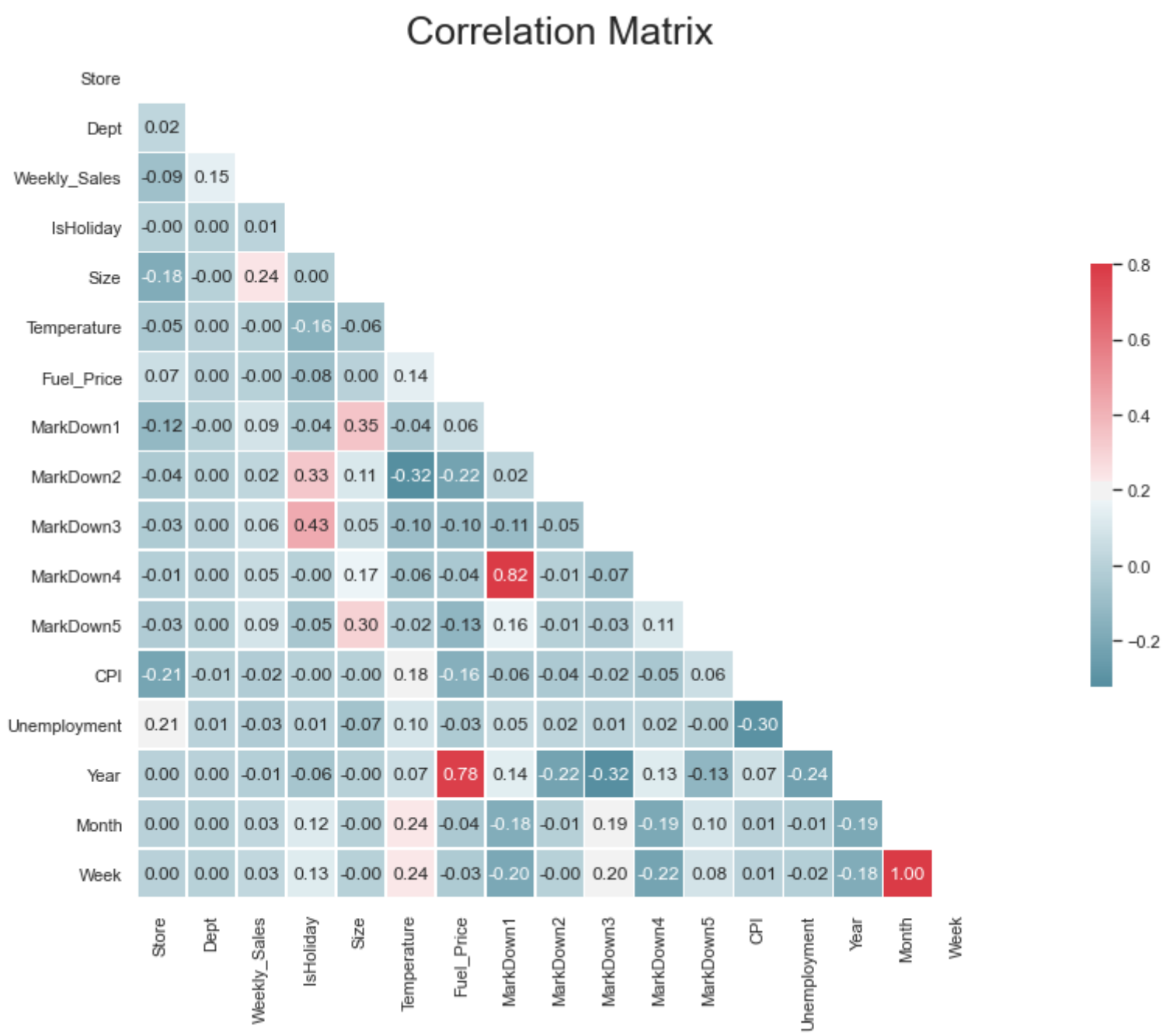
The Scatterplot Matrix (also known as SPLOM or Scatterplot Matrix) is used to roughly show the relationships between different columns in n-column data. It is possible to roughly estimate which variables are positively correlated and which are negatively correlated, thus providing decisions for the next data analysis. The scatter matrix diagram generated by Seaborn library is not a normal scatter matrix diagram, but the probability density distribution of the column is diagonally marked instead of the column name.

In [46]:
```
sns.set()
sns.pairplot(train_new[continuous_features],size = 2 ,kind ='scatter',diag_kind='kde')
plt.show()
```



The relationship between the weekly sales and features.

In [47]:
```
sns.set(style="white")

corr = train_new.corr()

mask = np.triu(np.ones_like(corr, dtype=np.bool))

f, ax = plt.subplots(figsize=(25, 10))

cmap = sns.diverging_palette(220, 10, as_cmap=True)

plt.title('Correlation Matrix', fontsize=25)

sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.8, center=0.2,
            square=True, linewidths=.5, cbar_kws={"shrink": .5},
            annot=True,fmt=".2f")

plt.show()
```



The following information found in the analysis of 'feature' dataset just now:

- Markdown1 and Markdown4 are highly correlated, so consider removing one.
- Except for Markdown5, there is a certain correlation between holiday and Markdowns, and Markdown3 has a high correlation with holiday.
- Only MarkDown1 has a strong negative correlation with Store, while other promotional activities have a weak correlation with Store.
- 'CPI' has a weak correlation with unemployment and promotional activity.
- The negative relationship between CPI and unemployment rate, conforms to economic logic.
- There is a negative correlation between CPI and store number. Can it be explained that the smaller (older) the store number is, the higher the regional consumption index (high consumption level) is?

We can also know from this figure:

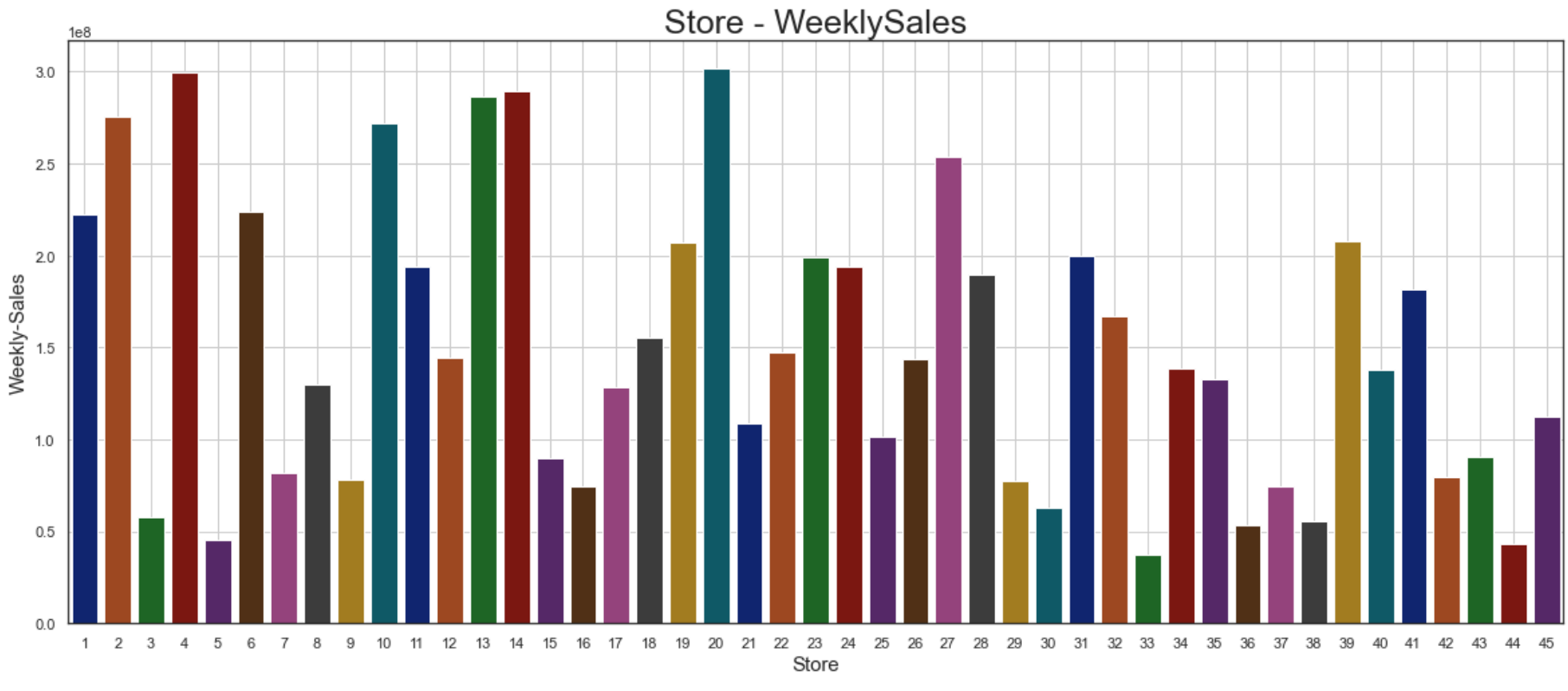
- The department number has no connection with any dimension.
- Weekly sales had a positive correlation with Size and Dept.
- To my surprise, the correlation between weekly sales and the five promotion activities is not very high.
- Size has a positive correlation with all five promotions. Does it mean that some promotions are usually available in stores of a specified Size?
- There was a higher negative correlation between temperature and MarkDown2. Interestingly, the graph of MarkDown2 changing over time showed that such promotion was probably only available during the cold months, as there was no such promotion between May and October.
- The strong positive correlation between fuel prices and annual prices seems to be consistent with the chart above, which shows oil prices getting more expensive each year.
- MarkDown2 and MarkDown3 have a certain relationship with 'Year' and are consistent with the information obtained from their images. Both promotions are rare and only available on designated days of the year.
- The unemployment rate is getting lower and lower.

Visualize the relationship between weekly sales and features.

Store - WeeklySales

By observing the graph of the change of weekly sales over time, it can be found that there is a gap in sales between stores on the same date. Look at the relationship between stores and weekly sales in detail.

In [48]:
```
# Figure out the corresponding total sales by grouping the 'Store' field
store_WS = train_new.groupby('Store')['Weekly_Sales'].sum()
plt.figure(figsize=(20,8))
sns.barplot(store_WS.index,store_WS.values,palette='dark')
plt.title('Store - WeeklySales',fontsize=25)
plt.xlabel('Store',fontsize=15)
plt.ylabel('Weekly-Sales',fontsize=15)
plt.grid()
plt.show()
```



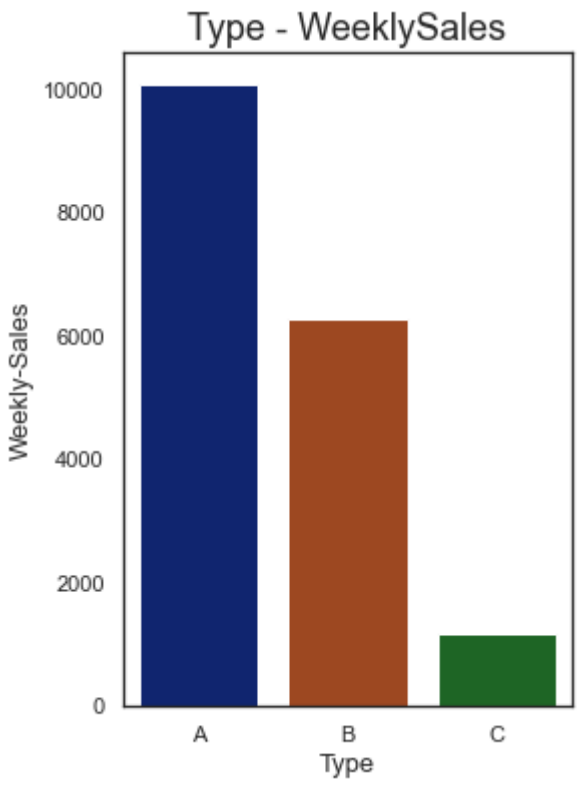
It can be seen that the weekly sales of some stores are very large, while the weekly sales of some stores are very small. Is weekly Sales related to the "Type","Size" or "Dept" of stores?

Type - WeeklySales

Group the "Type" field to calculate the corresponding weekly sales median

In [49]:
```
type_WS = train_new.groupby('Type')['Weekly_Sales'].median()
```

In [50]:
```
plt.figure(figsize=(4,6))
sns.barplot(type_WS.index,type_WS.values,palette='dark')
plt.title('Type - WeeklySales',fontsize=18)
plt.xlabel('Type',fontsize=13)
plt.ylabel('Weekly-Sales',fontsize=13)
plt.show()
```

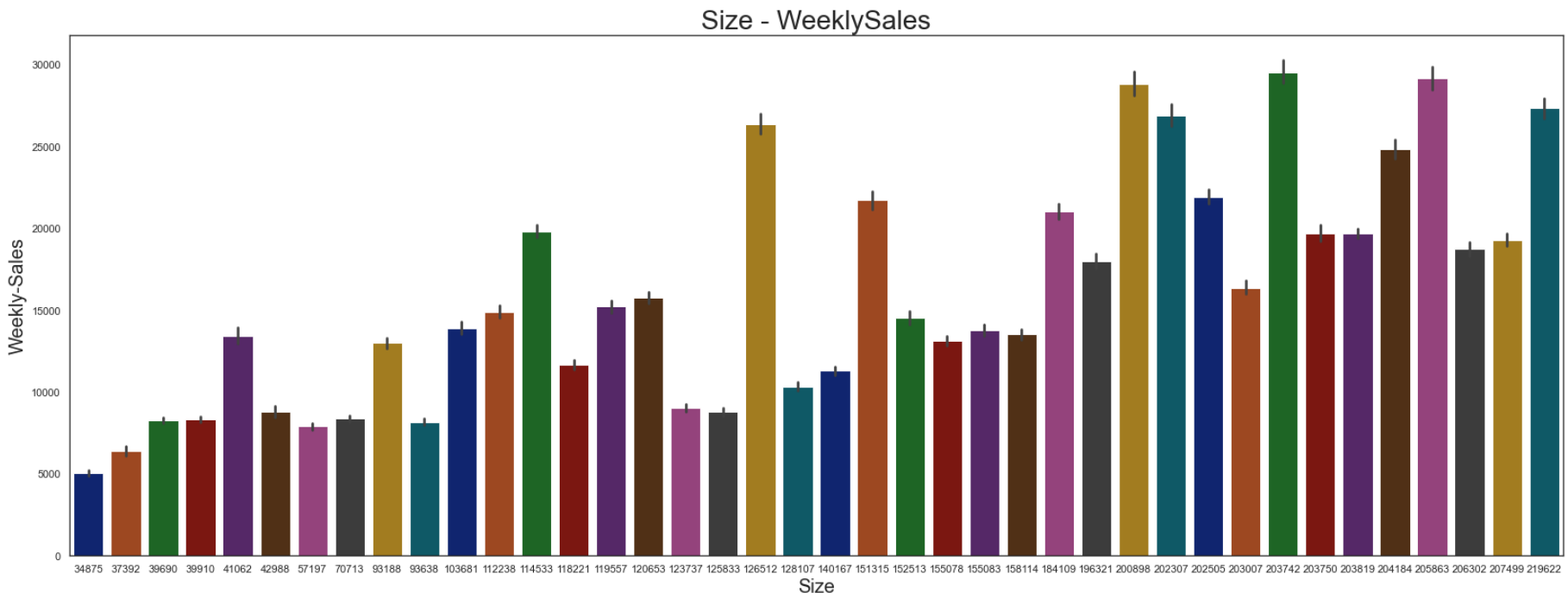


We can see that the median of weekly sales in Type A is much larger than the average of any other type of store. And type B stores are much bigger than Type C stores.

Size - WeeklySales

Although Size is not a Continuous variable, the number of stores is small, so in order to observe the relationship between them more clearly, histogram is used to display.

In [51]:
```
plt.figure(figsize=(28,10))
sns.barplot(x='Size',y='Weekly_Sales',data=train_new,palette='dark')
plt.title('Size - WeeklySales',fontsize=28)
plt.xlabel('Size',fontsize=20)
plt.ylabel('Weekly-Sales',fontsize=20)
plt.show()
```

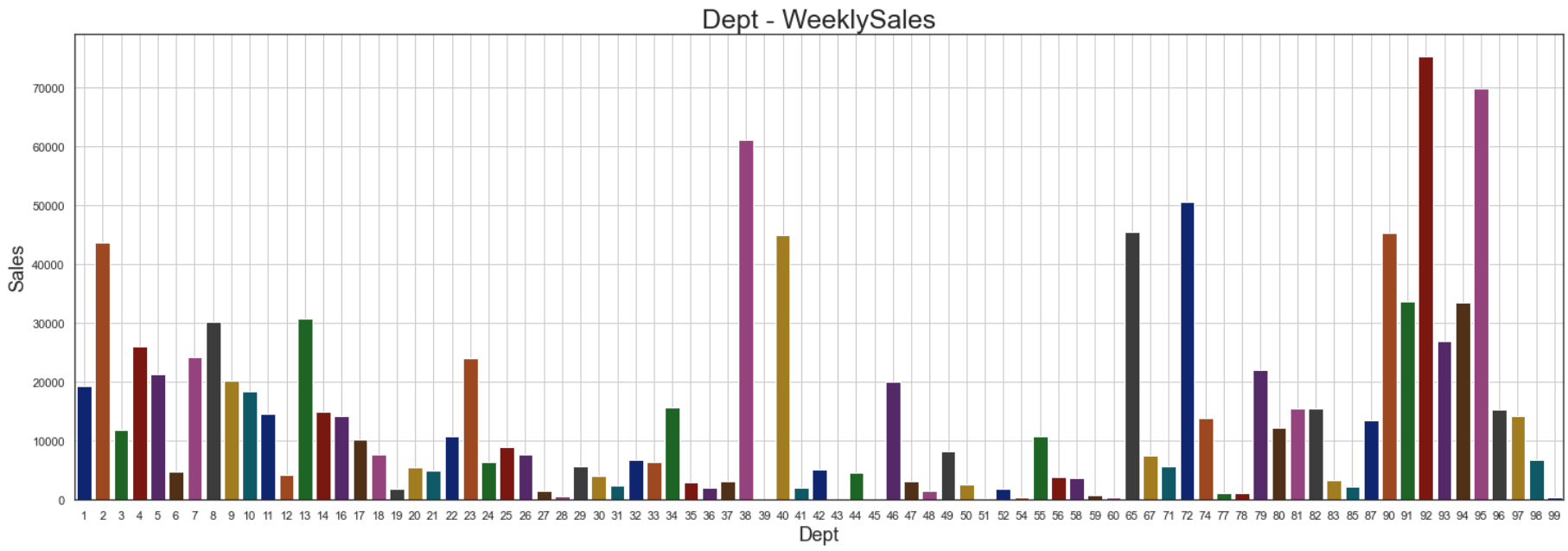


As can be seen, with the increase in the size of the store, weekly sales have a significant upward trend.

Dept - WeeklySales

Group the "Dept" field to calculate the means of the corresponding weekly sales

In [52]:
```
dept_WS = train_new.groupby('Dept')['Weekly_Sales'].mean()
plt.figure(figsize=(25,8))
sns.barplot(dept_WS.index, dept_WS.values, palette='dark')
plt.ylabel('Sales', fontsize=18)
plt.xlabel('Dept', fontsize=18)
plt.grid()
plt.title('Dept - WeeklySales',fontsize=25)
plt.show()
```



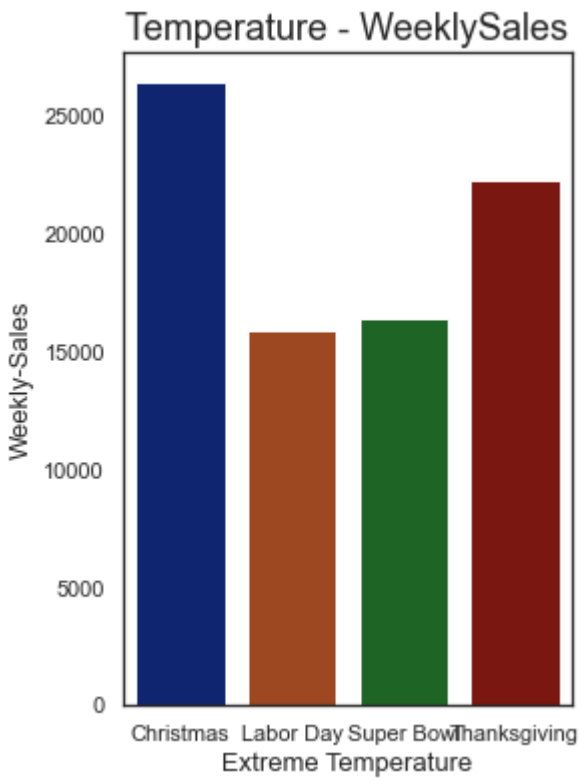
As we can see, there are seven departments with very high weekly sales. The department Numbers are 2,38,40,72,90,92,95.

Holiday - WeeklySales

Personally, I think that in general, people shop before Christmas, not after Christmas, so I need to modify the data of Christmas in the data set.

In [53]:
```
holidays = {
    'Super Bowl': pd.to_datetime(['12-Feb-10', '11-Feb-11', '10-Feb-12', '8-Feb-13']),
    'Labor Day': pd.to_datetime(['10-Sep-10', '9-Sep-11', '7-Sep-12', '6-Sep-13']),
    'Thanksgiving': pd.to_datetime(['26-Nov-10', '25-Nov-11', '23-Nov-12', '29-Nov-13']),
    'Christmas': pd.to_datetime(['24-Dec-10', '23-Dec-11', '21-Dec-12', '20-Dec-13'])
}

train_new.insert(train_new.columns.tolist().index('IsHoliday')+1, 'Holiday', np.nan)
for holiday in holidays:
    train_new.loc[train_new['Date'].isin(holidays[holiday]), 'Holiday'] = holiday
```

In [54]:
```
type_Tem = train_new.groupby('Holiday')['Weekly_Sales'].mean()
plt.figure(figsize=(4,6))
sns.barplot(type_Tem.index,type_Tem.values,palette='dark')
plt.title('Temperature - WeeklySales',fontsize=18)
plt.xlabel('Extreme Temperature',fontsize=13)
plt.ylabel('Weekly-Sales',fontsize=13)
plt.show()
```



Weekly sales are particularly high during the Christmas and Thanksgiving periods.

In [55]:
```
train_new = pd.get_dummies(train_new,columns=['Holiday'])
```

In [56]:
```
train_new.corr()['Weekly_Sales'].sort_values(ascending=False)
```

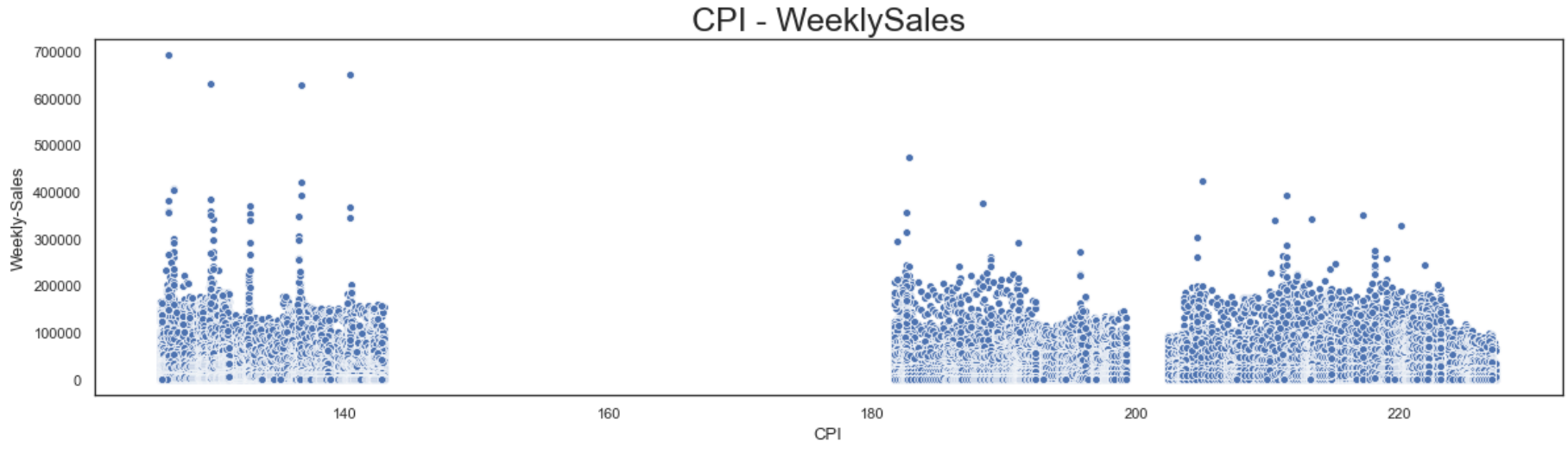
Out[56]:
```
Weekly_Sales            1.000000
Size                    0.243845
Dept                    0.148307
MarkDown5               0.090385
MarkDown1               0.085245
MarkDown3               0.060345
Holiday_Christmas       0.055013
MarkDown4               0.045362
Holiday_Thanksgiving    0.032881
Month                   0.028406
Week                    0.027668
MarkDown2               0.024303
IsHoliday               0.012806
Holiday_Super Bowl      0.002532
Fuel_Price             -0.000020
Holiday_Labor Day      -0.000637
Temperature            -0.002321
Year                   -0.010056
CPI                    -0.021021
Unemployment           -0.025837
Store                  -0.085111
Name: Weekly_Sales, dtype: float64
```

Get the following information:

- Labor Day and Super Bowl have almost zero correlation with weekly sales.
- CPI and unemployment rates are very low relative to weekly sales.

CPI - WeeklySales

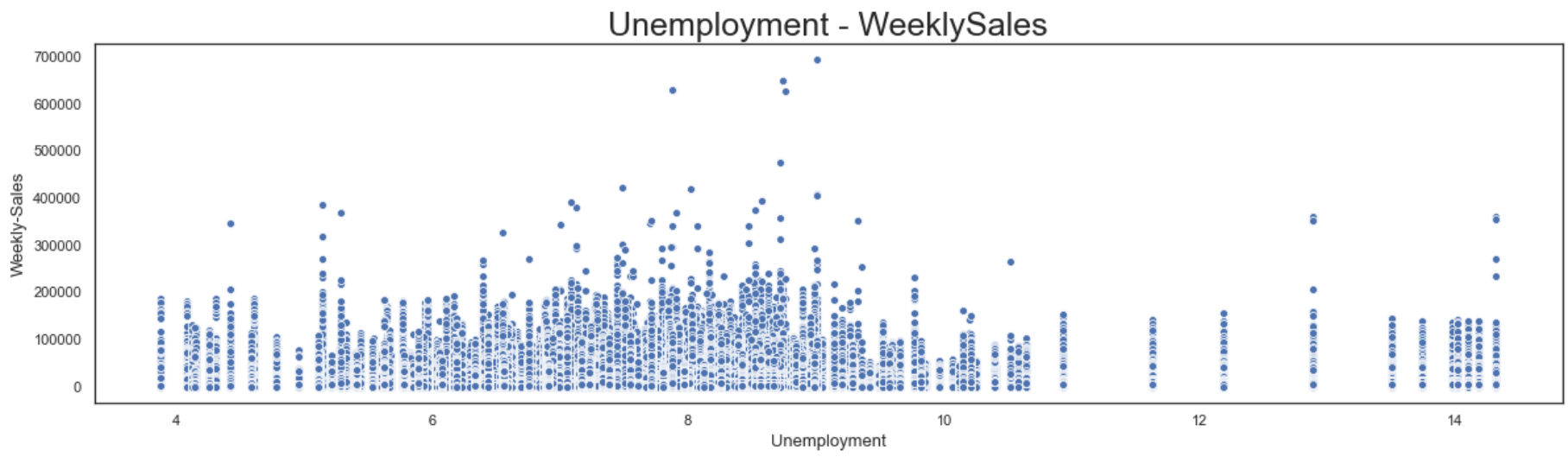
In [57]:
```
plt.figure(figsize=(20,5))
sns.scatterplot(x="CPI", y="Weekly_Sales", data=train_new)
plt.title('CPI - WeeklySales',fontsize=25)
plt.xlabel('CPI',fontsize=13)
plt.ylabel('Weekly-Sales',fontsize=13)
plt.show()
```



You can see a fuzzy downward trend, but it's not obvious.

Unemployment - WeeklySales

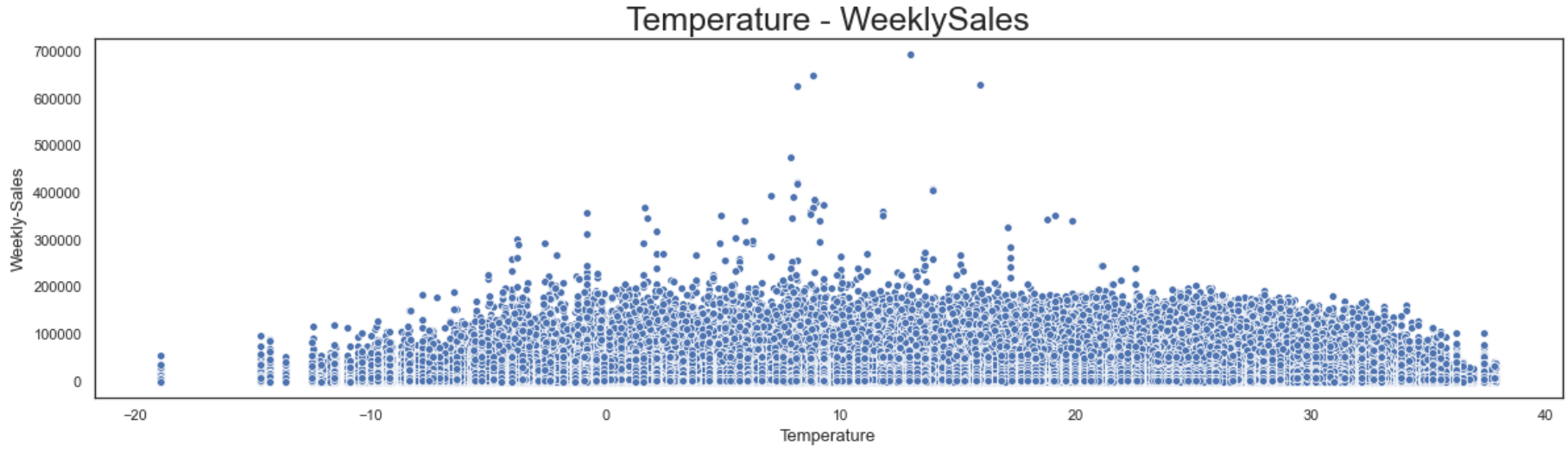
In [58]:
```
plt.figure(figsize=(20,5))
sns.scatterplot(x="Unemployment", y="Weekly_Sales", data=train_new)
plt.title('Unemployment - WeeklySales',fontsize=25)
plt.xlabel('Unemployment',fontsize=13)
plt.ylabel('Weekly-Sales',fontsize=13)
plt.show()
```



When the unemployment rate is in the 8-10 range, you can see a few high weekly sales, but overall there is no significant upward or downward trend.

Temperature - WeeklySales

In [59]:
```
plt.figure(figsize=(20,5))
sns.scatterplot(x="Temperature", y="Weekly_Sales", data=train_new)
plt.title('Temperature - WeeklySales',fontsize=25)
plt.xlabel('Temperature',fontsize=13)
plt.ylabel('Weekly-Sales',fontsize=13)
plt.show()
```



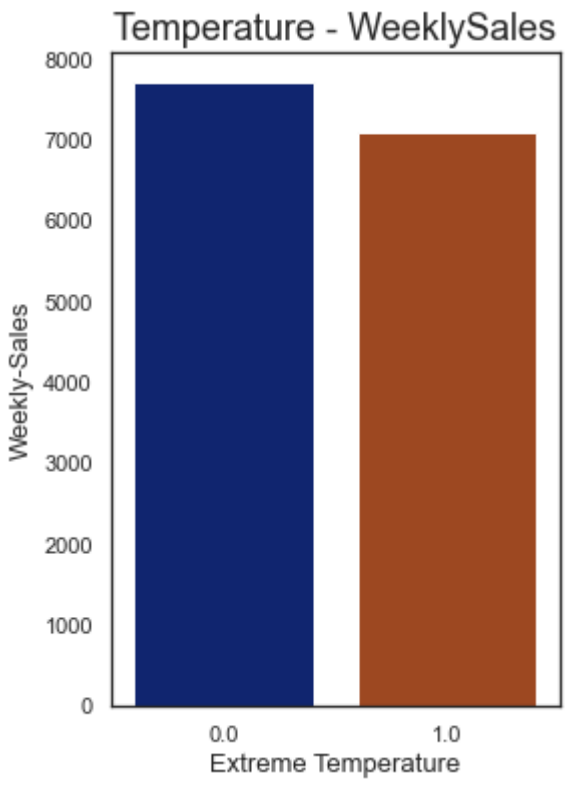
It is common for people not to want to go out in extreme temperature, with temperatures above 32 degrees Celsius and below zero degrees Celsius being used as thresholds for three groups

In [60]:
```
train_new.loc[(train_new["Temperature"]<0)|
              (train_new["Temperature"]>32), "Is_temp_extr"]=1
train_new.loc[(train_new["Temperature"]>=0)&
              (train_new["Temperature"]<=32), "Is_temp_extr"]=0


train_new.corr().Weekly_Sales.sort_values(ascending=False)[["Temperature", "Is_temp_extr"]]
```

Out[60]:
```
Temperature    -0.002321
Is_temp_extr   -0.016578
Name: Weekly_Sales, dtype: float64
```

By breaking down the numerical interval of the temperature dimension, we can see that the correlation between extreme weather and weekly sales increases significantly when extreme weather occurs.

In [61]:
```
type_Tem = train_new.groupby('Is_temp_extr')['Weekly_Sales'].median()
plt.figure(figsize=(4,6))
sns.barplot(type_Tem.index,type_Tem.values,palette='dark')
plt.title('Temperature - WeeklySales',fontsize=18)
plt.xlabel('Extreme Temperature',fontsize=13)
plt.ylabel('Weekly-Sales',fontsize=13)
plt.show()
```



The extreme temperatures reduced weekly sales by nearly a thousand dollars compared with normal temperatures.

Data preprocessing and feature engineering

Data preprocessing

- Data preprocessing is the process of detecting, correcting, or deleting corrupted, inaccurate, or inapplicable records from the data. It may face problems such as different types of data, such as text, Numbers, time series, continuity, and discontinuity. It could be that the quality of the data is poor, there is noise, there are exceptions, there are deletions, there are errors in the data, there are dimensions, there are duplications, the data is skewed, the amount of data is too large or too small.
- The purpose of data preprocessing: to adapt the data to the model and match the requirements of the model.

Feature Engineering

- Feature engineering is the process of transforming the original data into features that better represent the potential problems of the predictive model, which can be achieved by 1) selecting the most relevant features, 2) extracting features and 3) creating features. Possible problems include correlation between features, irrelevant features and labels, too many or too few features, or simply not showing the data as it should be.
- Purpose of feature engineering : 1) To reduce calculation cost and 2) to raise the upper limit of the model

Data preprocessing

Integration of the data

In [64]:
```
train = pd.read_csv('/Users/zhiyongmai/Desktop/Walmart/walmart-recruiting-store-sales-forecasting/train.csv',
parse_dates=['Date'])
test = pd.read_csv('/Users/zhiyongmai/Desktop/Walmart/walmart-recruiting-store-sales-forecasting/test.csv',pa
rse_dates=['Date'])
store = pd.read_csv('/Users/zhiyongmai/Desktop/Walmart/walmart-recruiting-store-sales-forecasting/stores.csv'
)
feature = pd.read_csv('/Users/zhiyongmai/Desktop/Walmart/walmart-recruiting-store-sales-forecasting/features.
csv',parse_dates=['Date'])
```

Display all dataset again

In [65]:
```
display_side_by_side(
    [datasets[k].head() for k in datasets],
    [k for k in datasets])
```

train

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05 00:00:00	24924.500000	False
1	1	1	2010-02-12 00:00:00	46039.490000	True
2	1	1	2010-02-19 00:00:00	41595.550000	False
3	1	1	2010-02-26 00:00:00	19403.540000	False
4	1	1	2010-03-05 00:00:00	21827.900000	False

test

	Store	Dept	Date	IsHoliday
0	1	1	2012-11-02 00:00:00	False
1	1	1	2012-11-09 00:00:00	False
2	1	1	2012-11-16 00:00:00	False
3	1	1	2012-11-23 00:00:00	True
4	1	1	2012-11-30 00:00:00	False

store

	Store	Type	Size
0	1	A	151315
1	2	A	202307
2	3	B	37392
3	4	A	205863
4	5	B	34875

features

	Store	Date	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5	CPI	Unemployment	IsHolida
0	1	2010-02-05 00:00:00	5.727778	2.572000	nan	nan	nan	nan	nan	211.096358	8.106000	Fals
1	1	2010-02-12 00:00:00	3.616667	2.548000	nan	nan	nan	nan	nan	211.242170	8.106000	Tru
2	1	2010-02-19 00:00:00	4.405556	2.514000	nan	nan	nan	nan	nan	211.289143	8.106000	Fals
3	1	2010-02-26 00:00:00	8.127778	2.561000	nan	nan	nan	nan	nan	211.319643	8.106000	Fals
4	1	2010-03-05 00:00:00	8.055556	2.625000	nan	nan	nan	nan	nan	211.350143	8.106000	Fals

In [66]:
```
train['Set'] = 'Train'
test['Set'] = 'Test'
```

In [67]:
```
df = pd.concat([train,test])
```

In [68]:
```
df = df.merge(store,how='left',on='Store').merge(feature,how='left',on=['Store','Date','IsHoliday'])
```

In [69]:
```
df = df.sort_values(by=['Set','Store','Dept','Date'],ascending=[False,True,True,True])
```

Information of 'df' dataset

In [70]:
```
df.shape
```

Out[70]: (536634, 17)

In [71]:
```
df.head()
```

Out[71]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Set	Type	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3	MarkDown4
0	1	1	2010-02-05	24924.50	False	Train	A	151315	42.31	2.572	NaN	NaN	NaN	NaN
1	1	1	2010-02-12	46039.49	True	Train	A	151315	38.51	2.548	NaN	NaN	NaN	NaN
2	1	1	2010-02-19	41595.55	False	Train	A	151315	39.93	2.514	NaN	NaN	NaN	NaN
3	1	1	2010-02-26	19403.54	False	Train	A	151315	46.63	2.561	NaN	NaN	NaN	NaN
4	1	1	2010-03-05	21827.90	False	Train	A	151315	46.50	2.625	NaN	NaN	NaN	NaN

In [72]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 536634 entries, 0 to 536633
Data columns (total 17 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   Store         536634 non-null  int64
 1   Dept          536634 non-null  int64
 2   Date          536634 non-null  datetime64[ns]
 3   Weekly_Sales  421570 non-null  float64
 4   IsHoliday     536634 non-null  bool
 5   Set           536634 non-null  object
 6   Type          536634 non-null  object
 7   Size          536634 non-null  int64
 8   Temperature   536634 non-null  float64
 9   Fuel_Price    536634 non-null  float64
 10  MarkDown1     265596 non-null  float64
 11  MarkDown2     197685 non-null  float64
 12  MarkDown3     242326 non-null  float64
 13  MarkDown4     237143 non-null  float64
 14  MarkDown5     266496 non-null  float64
 15  CPI           498472 non-null  float64
 16  Unemployment  498472 non-null  float64
dtypes: bool(1), datetime64[ns](1), float64(10), int64(3), object(2)
memory usage: 70.1+ MB
```

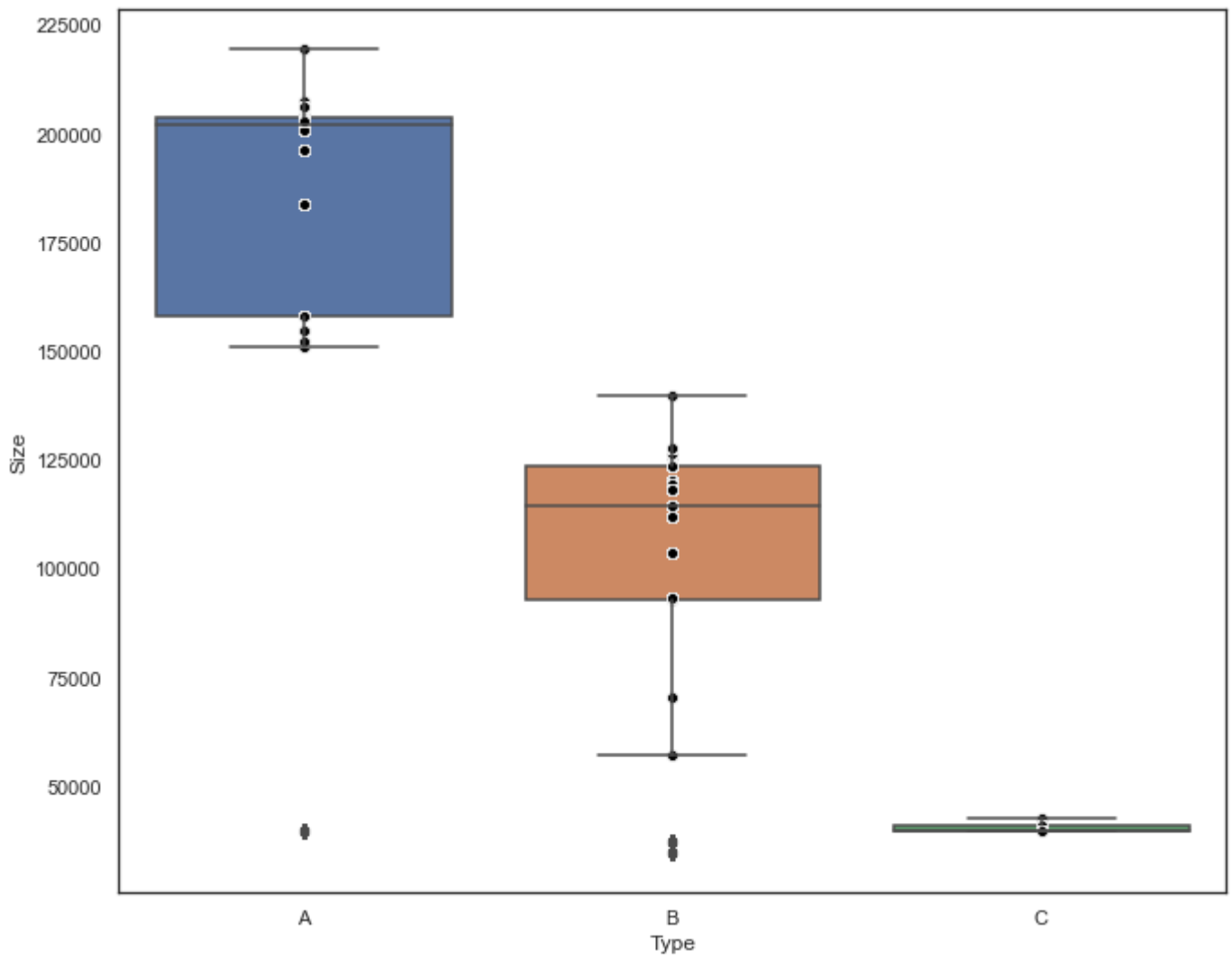
Handling outliers

Process the 'Size' field

Method: Use a boxplot to screen for exceptions and truncate (handle outliers).

In [73]:
```
sns.boxplot(x='Type',y='Size',data=df)
sns.scatterplot(x='Type',y='Size',data=df,color='black')
```

Out[73]: <AxesSubplot:xlabel='Type', ylabel='Size'>



In [74]:
```
sizeA = df.loc[df.loc[:,'Type']=='A','Size']
sizeB = df.loc[df.loc[:,'Type']=='B','Size']
sizeC = df.loc[df.loc[:,'Type']=='C','Size']
scale = 3
```

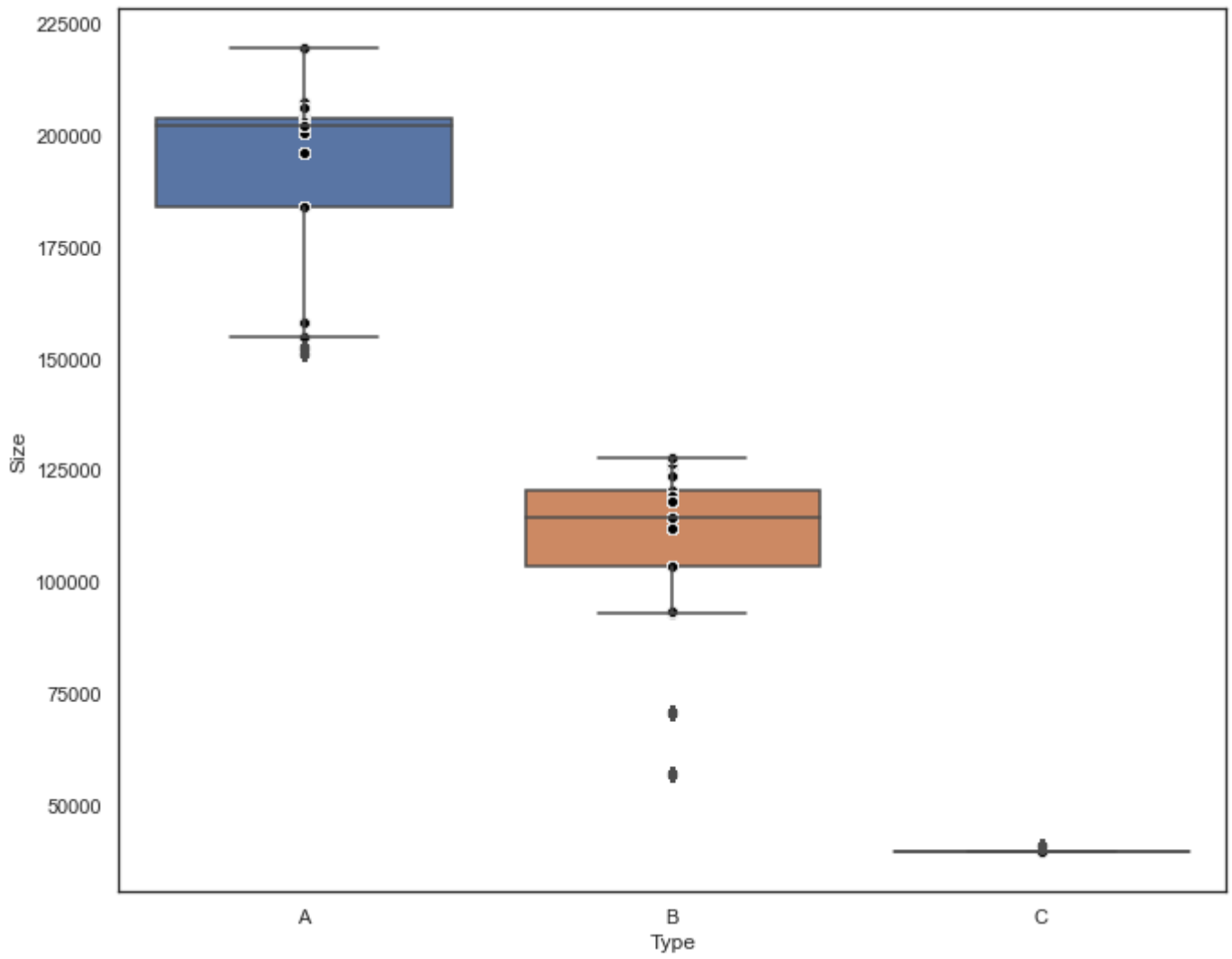
In [75]:
```
Q1_A = df.loc[df.loc[:,'Type']=='A','Size'].quantile(0.25)
Q3_A = df.loc[df.loc[:,'Type']=='A','Size'].quantile(0.75)
IQR_A = Q3_A - Q1_A
outlier_list_A = sizeA[(sizeA < Q1_A - (1.5 * IQR_A))|(sizeA > Q1_A + (1.5 * IQR_A))].index.to_list()
df['Size'].iloc[outlier_list_A]=df.groupby('Type')['Size'].median()['A']
```

In [76]:
```
Q1_B = df.loc[df.loc[:,'Type']=='B','Size'].quantile(0.25)
Q3_B = df.loc[df.loc[:,'Type']=='B','Size'].quantile(0.75)
IQR_B = Q3_B - Q1_B
outlier_list_B = sizeB[(sizeB < Q1_B - (1.5 * IQR_B))|(sizeB > Q1_B + (1.5 * IQR_B))].index.to_list()
df['Size'].iloc[outlier_list_B]=df.groupby('Type')['Size'].median()['B']
```

In [77]:
```
Q1_C = df.loc[df.loc[:,'Type']=='C','Size'].quantile(0.25)
Q3_C = df.loc[df.loc[:,'Type']=='C','Size'].quantile(0.75)
IQR_C = Q3_C - Q1_C
outlier_list_C = sizeC[(sizeC < Q1_C - (1.5 * IQR_C))|(sizeC > Q1_C + (1.5 * IQR_C))].index.to_list()
df['Size'].iloc[[outlier_list_C]]=df.groupby('Type')['Size'].median()['C']
```

In [78]:
```
sns.boxplot(x='Type',y='Size',data=df)
sns.scatterplot(x='Type',y='Size',data=df,color='black')
```

Out[78]: <AxesSubplot:xlabel='Type', ylabel='Size'>



After processing, the original outliers have been returned to normal.

Process the 'Weekly_Sales' field

Replace the outliers with the median.

In [79]:
```
sales_medians = df['Weekly_Sales'].median()
df['Weekly_Sales'] = df['Weekly_Sales'].apply(lambda x:np.where(x<0,sales_medians,x))
df['Weekly_Sales'] = df['Weekly_Sales'].apply(lambda x:np.where(x==0,1,x))
df.describe()
```

Out[79]:

	Store	Dept	Weekly_Sales	Size	Temperature	Fuel_Price	MarkDown1	MarkDown2	MarkDown3
count	536634.000000	536634.000000	421570.000000	536634.000000	536634.000000	536634.000000	265596.000000	197685.000000	242326.000000
mean	22.208621	44.277301	16004.669877	144302.704709	58.771762	3.408310	7438.004144	3509.274827	1857.913525
std	12.790580	30.527358	22698.578122	55864.110001	18.678716	0.430861	9411.341379	8992.047197	11616.143274
min	1.000000	1.000000	0.010000	39690.000000	-7.290000	2.472000	-2781.450000	-265.760000	-179.260000
25%	11.000000	18.000000	2130.877500	114533.000000	45.250000	3.041000	2114.640000	72.500000	7.220000
50%	22.000000	37.000000	7612.030000	151315.000000	60.060000	3.523000	5126.540000	385.310000	40.760000
75%	33.000000	74.000000	20205.852500	202505.000000	73.230000	3.744000	9303.850000	2392.390000	174.260000
max	45.000000	99.000000	693099.360000	219622.000000	101.950000	4.468000	103184.980000	104519.540000	149483.310000

The minimum value of 'Weekly_Sales' becomes 0.01 and is no longer negative.

Process 'MarkDown1-5' fields

MarkDown1-5 has negative values. The approach here is the same as for processing outliers in weekly sales.

In [80]:
```
#MarkDown1
MarkDown1_medians = df['MarkDown1'].median()
df['MarkDown1'] = df['MarkDown1'].apply(lambda x:np.where(x<0,MarkDown1_medians,x))

#MarkDown2
MarkDown2_medians = df['MarkDown2'].median()
df['MarkDown2'] = df['MarkDown2'].apply(lambda x:np.where(x<0,MarkDown2_medians,x))

#MarkDown3
MarkDown3_medians = df['MarkDown3'].median()
df['MarkDown3'] = df['MarkDown3'].apply(lambda x:np.where(x<0,MarkDown3_medians,x))

#MarkDown4
MarkDown4_medians = df['MarkDown4'].median()
df['MarkDown4'] = df['MarkDown4'].apply(lambda x:np.where(x<0,MarkDown4_medians,x))

#MarkDown5
MarkDown5_medians = df['MarkDown5'].median()
df['MarkDown5'] = df['MarkDown5'].apply(lambda x:np.where(x<0,MarkDown5_medians,x))

df.describe()[['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5']]
```

Out[80]:

	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5
count	265596.000000	197685.000000	242326.000000	237143.000000	266496.000000
mean	7442.799733	3512.858472	1858.164353	3371.556866	4325.540175
std	9408.550924	8990.717553	11616.102908	6872.281734	13548.932170
min	0.270000	0.000000	0.000000	0.220000	40.980000
25%	2128.980000	77.840000	7.350000	336.240000	1570.410000
50%	5126.540000	385.310000	40.760000	1239.040000	2870.910000
75%	9303.850000	2392.390000	174.260000	3397.080000	5012.220000
max	103184.980000	104519.540000	149483.310000	67474.850000	771448.100000

> By processing, the minimum values of the promotion activities are all greater than or equal to 0.

View and process missing values

The data used in machine learning and data mining can never be perfect. Many features are of great significance for analysis and modeling, but not for real data collectors. Therefore, in data mining, there are often important fields with many missing values, but fields cannot be abandoned. Therefore, one of the most important aspects of data preprocessing is missing values.

In [81]:
```
print('The number of missing values for each feature:')
df.isna().sum()
```

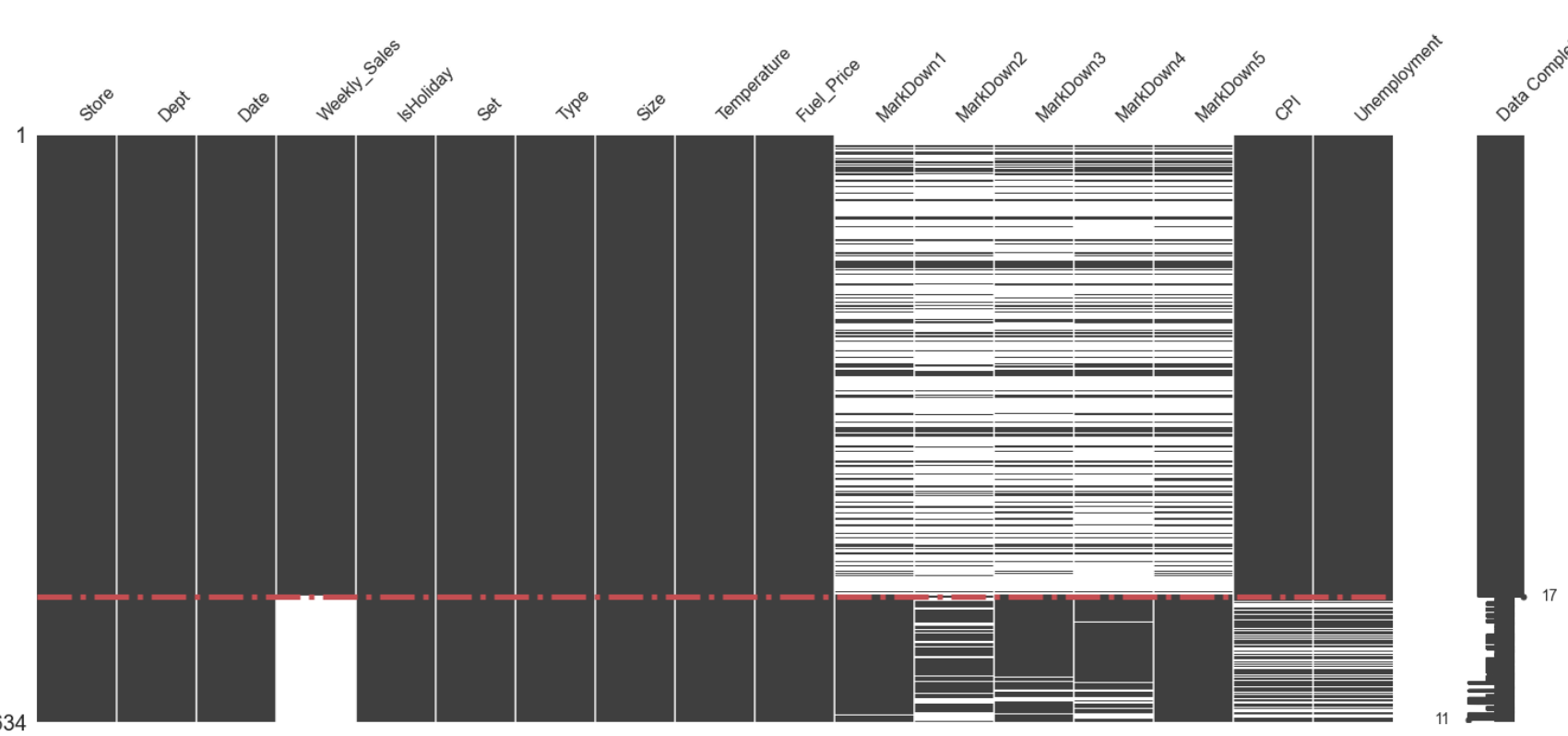
```
The number of missing values for each feature:
```

Out[81]:
```
Store              0
Dept               0
Date               0
Weekly_Sales    115064
IsHoliday          0
Set                0
Type               0
Size               0
Temperature        0
Fuel_Price         0
MarkDown1       271038
MarkDown2       338949
MarkDown3       294308
MarkDown4       299491
MarkDown5       270138
CPI              38162
Unemployment     38162
dtype: int64
```

Visualize missing data

In [82]:
```
plt.figure(figsize=(12,8))
l = df[df.Set=='Train'].tail(1).index[0]
f = msno.matrix(df,labels=True)
f.axhline(l, ls='-.', color='r',linewidth=5)
```

Out[82]:
```
<matplotlib.lines.Line2D at 0x7faad2938d10>

<Figure size 864x576 with 0 Axes>
```



The following information is obtained:

- The number of missing values of 'Markdown' is very large, and most of the missing values are above the red line (Train data set). However, from the analysis in the third part, it is known that 'Markdown' is highly correlated with 'IsHoliday', 'Store' ,'Department' and 'Temperature', so this four variables can be used to fill in the missing values of Markdown.
- Both the CPI and Unemployment missing values are concentrated below the red line (the test data set). The analysis in Part 3 shows that CPI and Unemployment have a high correlation with 'Store' and 'Temperature', so these two variables can be used to fill in the missing values.
- The missing values for Weekly_Sales are concentrated below the red line (test dataset) because the test dataset is used for prediction, so it is normal for Weekly_Sales in the test dataset to be completely missing.

Process missing values in MarkDown1-5

Methods:

1. Create a copy of dataset nemed 'df' .
2. Group by 'Store', 'Dept', 'Temperature'and 'IsHoliday' to evaluate the median value of the promotion activities.
3. The missing value of the promotion activities is then backfilled.
4. Change the name of the promotion activities in 'df_copy' and see the missing values for the 'df_copy'.
5. Merge 'df_copy' with 'df'.
6. Fill in the missing value of the promotion field in 'df' with 'df_copy'.

In [83]:
```
df_copy = df.copy()
df_copy = df_copy.groupby(['Store','Dept','Temperature','IsHoliday']).median()[[
    'MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5']].reset_index()
df_copy = df_copy.fillna(method='bfill')
df_copy.head()
```

Out[83]:

	Store	Dept	Temperature	IsHoliday	MarkDown1	MarkDown2	MarkDown3	MarkDown4	MarkDown5
0	1	1	35.40	False	1214.08	25366.33	15.01	72.36	3940.02
1	1	1	36.39	True	1214.08	25366.33	15.01	72.36	3940.02
2	1	1	38.51	True	1214.08	25366.33	15.01	72.36	3940.02
3	1	1	39.93	False	1214.08	25366.33	15.01	72.36	3940.02
4	1	1	41.73	False	1214.08	25366.33	15.01	72.36	3940.02

In [84]:
```
df_copy.rename(columns={'MarkDown1':'MD1','MarkDown2':'MD2','MarkDown3':'MD3','MarkDown4':'MD4','MarkDown5':
'MD5'},inplace=True)
df_copy.isna().sum()
```

Out[84]:
```
Store           0
Dept            0
Temperature     0
IsHoliday       0
MD1             0
MD2             0
MD3             0
MD4             0
MD5             0
dtype: int64
```

In [85]:
```
df = df.merge(df_copy,on=['Store','Dept','Temperature','IsHoliday'],how='left')
```

In [86]:
```
df.MarkDown1.fillna(df['MD1'],inplace=True)
df.MarkDown2.fillna(df['MD2'],inplace=True)
df.MarkDown3.fillna(df['MD3'],inplace=True)
df.MarkDown4.fillna(df['MD4'],inplace=True)
df.MarkDown5.fillna(df['MD5'],inplace=True)
df.drop(columns=['MD1','MD2','MD3','MD4','MD5'],inplace=True)
df.isna().sum()
```

Out[86]:
```
Store              0
Dept               0
Date               0
Weekly_Sales    115064
IsHoliday            0
Set                  0
Type                 0
Size                 0
Temperature          0
Fuel_Price           0
MarkDown1            0
MarkDown2            0
MarkDown3            0
MarkDown4            0
MarkDown5            0
CPI              38162
Unemployment     38162
dtype: int64
```

After filling, markdown1-5 has no missing values.

Process missing values in 'CPI' and 'Unemployment'

Method: Same as in the previous step when dealing with promotion activities.

1. Group by 'Store', 'Temperature'and to evaluate the median value of 'CPI' and 'Unemployment'.
2. The missing values of the two features are then filled in by interpolation.

In [87]:
```
df_copy1 = df.groupby(['Store','Temperature']).median()[['CPI','Unemployment']].reset_index()
```

In [88]:
```
df_copy1['CPI'] = df_copy1['CPI'].interpolate()
df_copy1['Unemployment'] = df_copy1['Unemployment'].interpolate()
```

In [89]:
```
df_copy1.rename(columns={'CPI':'C','Unemployment':'U'},inplace=True)
df_copy1.isna().sum()
```

Out[89]:
```
Store          0
Temperature    0
C              0
U              0
dtype: int64
```

In [90]:
```
df = df.merge(df_copy1,on=['Store','Temperature'],how='left')
```

In [91]:
```
df.CPI.fillna(df['C'],inplace=True)
df.Unemployment.fillna(df['U'],inplace=True)
df.drop(columns=['C','U'],inplace=True)
df.isna().sum()
```

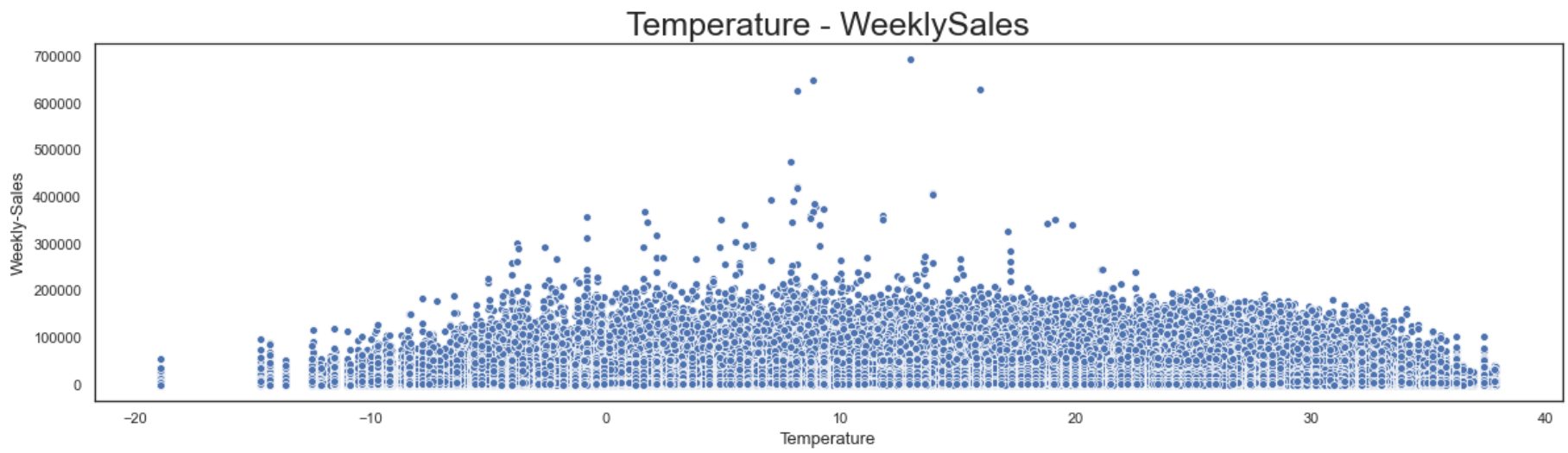
Out[91]:
```
Store              0
Dept               0
Date               0
Weekly_Sales    115064
IsHoliday          0
Set                0
Type               0
Size               0
Temperature        0
Fuel_Price         0
MarkDown1          0
MarkDown2          0
MarkDown3          0
MarkDown4          0
MarkDown5          0
CPI                0
Unemployment       0
dtype: int64
```

After filling, 'CPI' and 'Unemployment' has no missing values.

Data partition and feature creation

Data partition

In [92]:
```
df['Temperature'] = (df['Temperature']-32)*5/9
plt.figure(figsize=(20,5))
sns.scatterplot(x="Temperature", y="Weekly_Sales", data=df)
plt.title('Temperature - WeeklySales',fontsize=25)
plt.xlabel('Temperature',fontsize=13)
plt.ylabel('Weekly-Sales',fontsize=13)
plt.show()
```



According to the above figure, the following processing can be done:

- The temperature below -5°C is classified as extremely cold;
- The temperature between -5°C and 7°C is classified as cold.
- The temperature between 7°C and 20°C is classified as comfortable;

1. The Temperature between 20°C and 36°C degrees Celsius are classified as hot;
2. The Temperature above 36°C degrees Celsius are classified as very hot.

In [93]:
```
Temperature_Category = []
for i in df["Temperature"]:
    if i<=-5:
        Temperature_Category.append('ExtremelyCold')
    elif i>-5 and i<=7:
        Temperature_Category.append('Cold')
    elif i>7 and i<=20:
        Temperature_Category.append('Comfortable')
    elif i>20 and i<=36:
        Temperature_Category.append('Hot')
    else:
        Temperature_Category.append('ExtremelyHot')
```

In [94]:
```
df['Temperature_Category']=Temperature_Category
```

In [95]:
```
df.groupby('Temperature_Category')['Store'].count()
```

Out[95]:
```
Temperature_Category
Cold              113914
Comfortable       216590
ExtremelyCold      15507
ExtremelyHot        1106
Hot               189517
Name: Store, dtype: int64
```

feature creation

Mean, median,sum and standard deviation of promotion activity

In [96]:
```
markdown_features = ['MarkDown' + str(i) for i in range(1,6)]
df['markdown_sum'] = df[markdown_features].apply(lambda x: x.sum(), axis=1)
df['markdown_mean'] = df[markdown_features].apply(lambda x: x.mean(), axis=1)
df['markdown_std'] = df[markdown_features].apply(lambda x: x.std(), axis=1)
df['markdown_median'] = df[markdown_features].apply(lambda x: x.median(), axis=1)
```

In [97]:
```
df.head()
```

Out[97]:

	Store	Dept	Date	Weekly_Sales	IsHoliday	Set	Type	Size	Temperature	Fuel_Price	...	MarkDown3	MarkDown4	MarkDown5	C
0	1	1	2010-02-05	24924.50	False	Train	A	151315	5.727778	2.572	...	3.88	246.62	1900.40	211.0963
1	1	1	2010-02-12	46039.49	True	Train	A	151315	3.616667	2.548	...	15.01	72.36	3940.02	211.2421
2	1	1	2010-02-19	41595.55	False	Train	A	151315	4.405556	2.514	...	15.01	72.36	3940.02	211.2891
3	1	1	2010-02-26	19403.54	False	Train	A	151315	8.127778	2.561	...	634.70	24.90	2739.43	211.3196
4	1	1	2010-03-05	21827.90	False	Train	A	151315	8.055556	2.625	...	634.70	24.90	2739.43	211.3501

5 rows × 22 columns

Data conversion

The features to be processed here include:

Categorical features

- Temperature_Category: One-hot encoding conversion.

1. IsHoliday: Separate out all four holidays.One-hot encoding conversion.
2. Date: Sperate to days, months and years.
3. Type: Divide the dimension into three (A,B,C).One-hot encoding conversion.

When we are working with dummy variables, we need only (n-1) dummy variables. For instance, if we are working with dummies to treat month, we can not create dummies for all months. We need to pick one month and excluded it. This month ‘excluded’ will be our ‘control’. Why? Because, when all month’s dummy is zero, the remain value is related to our control variable.

Numeric features

- 'Size'

1. 'Temperature'
2. 'Fuel_Price'
3. 'MarkDown1'
4. 'MarkDown2'
5. 'MarkDown3'
6. 'MarkDown4'
7. 'MarkDown5'
8. 'CPI'
9. 'Unemployment'
10. 'markdown_sum'
11. 'markdown_mean'
12. 'markdown_std'
13. 'markdown_median'

In the analysis phase, we see that many of these numerical features are unevenly distributed, and we log them here.

Categorical features

1. Process the 'Temperature_Category' feature

In [98]:
```
df = pd.get_dummies(df,columns=['Temperature_Category'])
```

In [99]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 536634 entries, 0 to 536633
Data columns (total 26 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   Store                                 536634 non-null  int64
 1   Dept                                  536634 non-null  int64
 2   Date                                  536634 non-null  datetime64[ns]
 3   Weekly_Sales                          421570 non-null  float64
 4   IsHoliday                             536634 non-null  bool
 5   Set                                   536634 non-null  object
 6   Type                                  536634 non-null  object
 7   Size                                  536634 non-null  int64
 8   Temperature                           536634 non-null  float64
 9   Fuel_Price                            536634 non-null  float64
 10  MarkDown1                             536634 non-null  float64
 11  MarkDown2                             536634 non-null  float64
 12  MarkDown3                             536634 non-null  float64
 13  MarkDown4                             536634 non-null  float64
 14  MarkDown5                             536634 non-null  float64
 15  CPI                                   536634 non-null  float64
 16  Unemployment                          536634 non-null  float64
 17  markdown_sum                          536634 non-null  float64
 18  markdown_mean                         536634 non-null  float64
 19  markdown_std                          536634 non-null  float64
 20  markdown_median                       536634 non-null  float64
 21  Temperature_Category_Cold             536634 non-null  uint8
 22  Temperature_Category_Comfortable      536634 non-null  uint8
 23  Temperature_Category_ExtremelyCold    536634 non-null  uint8
 24  Temperature_Category_ExtremelyHot     536634 non-null  uint8
 25  Temperature_Category_Hot              536634 non-null  uint8
dtypes: bool(1), datetime64[ns](1), float64(14), int64(3), object(2), uint8(5)
memory usage: 89.0+ MB
```

2. Process the 'IsHoliday' field

In [100]:
```
holidays = {
    'Super Bowl': pd.to_datetime(['12-Feb-10', '11-Feb-11', '10-Feb-12', '8-Feb-13']),
    'Labor Day': pd.to_datetime(['10-Sep-10', '9-Sep-11', '7-Sep-12', '6-Sep-13']),
    'Thanksgiving': pd.to_datetime(['26-Nov-10', '25-Nov-11', '23-Nov-12', '29-Nov-13']),
    'Christmas': pd.to_datetime(['24-Dec-10', '23-Dec-11', '21-Dec-12', '20-Dec-13'])
}

df['Holiday'] = np.nan
for holiday in holidays:
    df.loc[df['Date'].isin(holidays[holiday]), 'Holiday'] = holiday
df = pd.get_dummies(df,columns=['Holiday'])
```

In [101]:
```
df.drop(columns='IsHoliday',inplace=True)
df['IsHoliday'] = np.nan
for holiday in holidays:
    df.loc[df['Date'].isin(holidays[holiday]), 'IsHoliday'] = 1
df['IsHoliday'] = df['IsHoliday'].fillna(0)
```

In [102]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 536634 entries, 0 to 536633
Data columns (total 30 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   Store                                 536634 non-null  int64
 1   Dept                                  536634 non-null  int64
 2   Date                                  536634 non-null  datetime64[ns]
 3   Weekly_Sales                          421570 non-null  float64
 4   Set                                   536634 non-null  object
 5   Type                                  536634 non-null  object
 6   Size                                  536634 non-null  int64
 7   Temperature                           536634 non-null  float64
 8   Fuel_Price                            536634 non-null  float64
 9   MarkDown1                             536634 non-null  float64
 10  MarkDown2                             536634 non-null  float64
 11  MarkDown3                             536634 non-null  float64
 12  MarkDown4                             536634 non-null  float64
 13  MarkDown5                             536634 non-null  float64
 14  CPI                                   536634 non-null  float64
 15  Unemployment                          536634 non-null  float64
 16  markdown_sum                          536634 non-null  float64
 17  markdown_mean                         536634 non-null  float64
 18  markdown_std                          536634 non-null  float64
 19  markdown_median                       536634 non-null  float64
 20  Temperature_Category_Cold             536634 non-null  uint8
 21  Temperature_Category_Comfortable      536634 non-null  uint8
 22  Temperature_Category_ExtremelyCold    536634 non-null  uint8
 23  Temperature_Category_ExtremelyHot     536634 non-null  uint8
 24  Temperature_Category_Hot              536634 non-null  uint8
 25  Holiday_Christmas                     536634 non-null  uint8
 26  Holiday_Labor Day                     536634 non-null  uint8
 27  Holiday_Super Bowl                    536634 non-null  uint8
 28  Holiday_Thanksgiving                  536634 non-null  uint8
 29  IsHoliday                             536634 non-null  float64
dtypes: datetime64[ns](1), float64(15), int64(3), object(2), uint8(9)
memory usage: 94.7+ MB
```

3. Processing 'date' feature

In [103]:
```
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Week'] = df['Date'].dt.week
# df = df.drop(columns='Date')
```

4 . Process the 'Type' field

In [104]:
```
df = pd.get_dummies(df,columns = ['Type'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 536634 entries, 0 to 536633
Data columns (total 35 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   Store                                 536634 non-null  int64
 1   Dept                                  536634 non-null  int64
 2   Date                                  536634 non-null  datetime64[ns]
 3   Weekly_Sales                          421570 non-null  float64
 4   Set                                   536634 non-null  object
 5   Size                                  536634 non-null  int64
 6   Temperature                           536634 non-null  float64
 7   Fuel_Price                            536634 non-null  float64
 8   MarkDown1                             536634 non-null  float64
 9   MarkDown2                             536634 non-null  float64
 10  MarkDown3                             536634 non-null  float64
 11  MarkDown4                             536634 non-null  float64
 12  MarkDown5                             536634 non-null  float64
 13  CPI                                   536634 non-null  float64
 14  Unemployment                          536634 non-null  float64
 15  markdown_sum                          536634 non-null  float64
 16  markdown_mean                         536634 non-null  float64
 17  markdown_std                          536634 non-null  float64
 18  markdown_median                       536634 non-null  float64
 19  Temperature_Category_Cold             536634 non-null  uint8
 20  Temperature_Category_Comfortable      536634 non-null  uint8
 21  Temperature_Category_ExtremelyCold    536634 non-null  uint8
 22  Temperature_Category_ExtremelyHot     536634 non-null  uint8
 23  Temperature_Category_Hot              536634 non-null  uint8
 24  Holiday_Christmas                     536634 non-null  uint8
 25  Holiday_Labor Day                     536634 non-null  uint8
 26  Holiday_Super Bowl                    536634 non-null  uint8
 27  Holiday_Thanksgiving                  536634 non-null  uint8
 28  IsHoliday                             536634 non-null  float64
 29  Year                                  536634 non-null  int64
 30  Month                                 536634 non-null  int64
 31  Week                                  536634 non-null  int64
 32  Type_A                                536634 non-null  uint8
 33  Type_B                                536634 non-null  uint8
 34  Type_C                                536634 non-null  uint8
dtypes: datetime64[ns](1), float64(15), int64(6), object(1), uint8(12)
memory usage: 104.4+ MB
```

In [105]:
```
df.to_csv('df.csv',index=False)
```

Numeric features

In [106]:
```
numeric_features = ['Size','Temperature','Fuel_Price', 'MarkDown1', 'MarkDown2', 'MarkDown3',
       'MarkDown4', 'MarkDown5', 'CPI', 'Unemployment', 'markdown_sum',
       'markdown_mean', 'markdown_std', 'markdown_median']
```

1. Box - Cox processing

Box - Cox transform method to solve error term in regression model is not obey gaussian distribution violation problem. Usually the violation occurs, the error ϵ prediction variables related, will influence the accuracy of the model results. This method can effectively make the data more normality.

In [107]:
```
for col in df[numeric_features].columns:
    print('{:15}'.format(col),
          'Skewness: {:05.2f}'.format(df[col].skew()) ,
          '   ' ,
          'Kurtosis: {:06.2f}'.format(df[col].kurt())
         )
```

```
Size            Skewness: -0.39     Kurtosis: -00.99
Temperature     Skewness: -0.27     Kurtosis: -00.63
Fuel_Price      Skewness: -0.31     Kurtosis: -00.96
MarkDown1       Skewness: 04.21     Kurtosis: 025.79
MarkDown2       Skewness: 05.04     Kurtosis: 032.08
MarkDown3       Skewness: 08.59     Kurtosis: 079.79
MarkDown4       Skewness: 05.18     Kurtosis: 032.64
MarkDown5       Skewness: 55.85     Kurtosis: 4088.49
CPI             Skewness: 00.09     Kurtosis: -01.82
Unemployment    Skewness: 01.07     Kurtosis: 002.60
markdown_sum    Skewness: 07.12     Kurtosis: 168.75
markdown_mean   Skewness: 07.12     Kurtosis: 168.75
markdown_std    Skewness: 12.95     Kurtosis: 460.13
markdown_median Skewness: 02.63     Kurtosis: 017.90
```

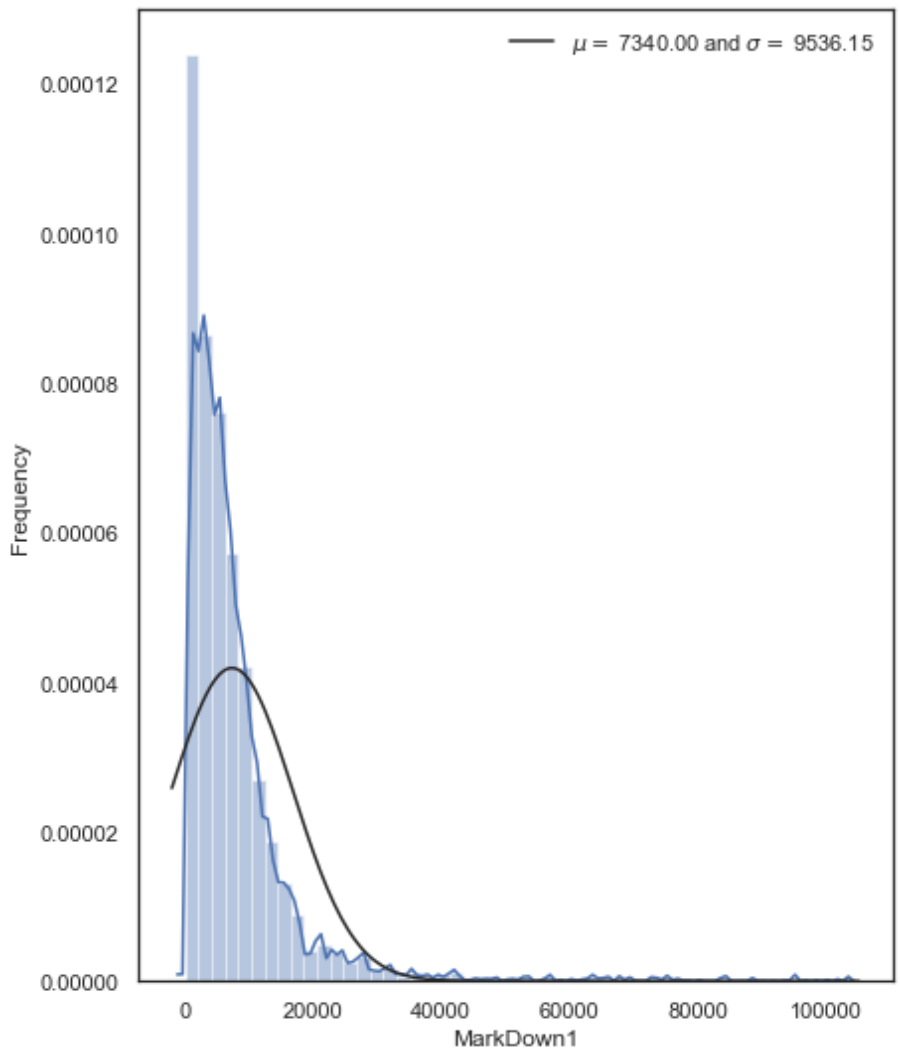
The more the skewness and kurtosis deviate from zero, the worse the model learning will be. The features that need to be normalised are:

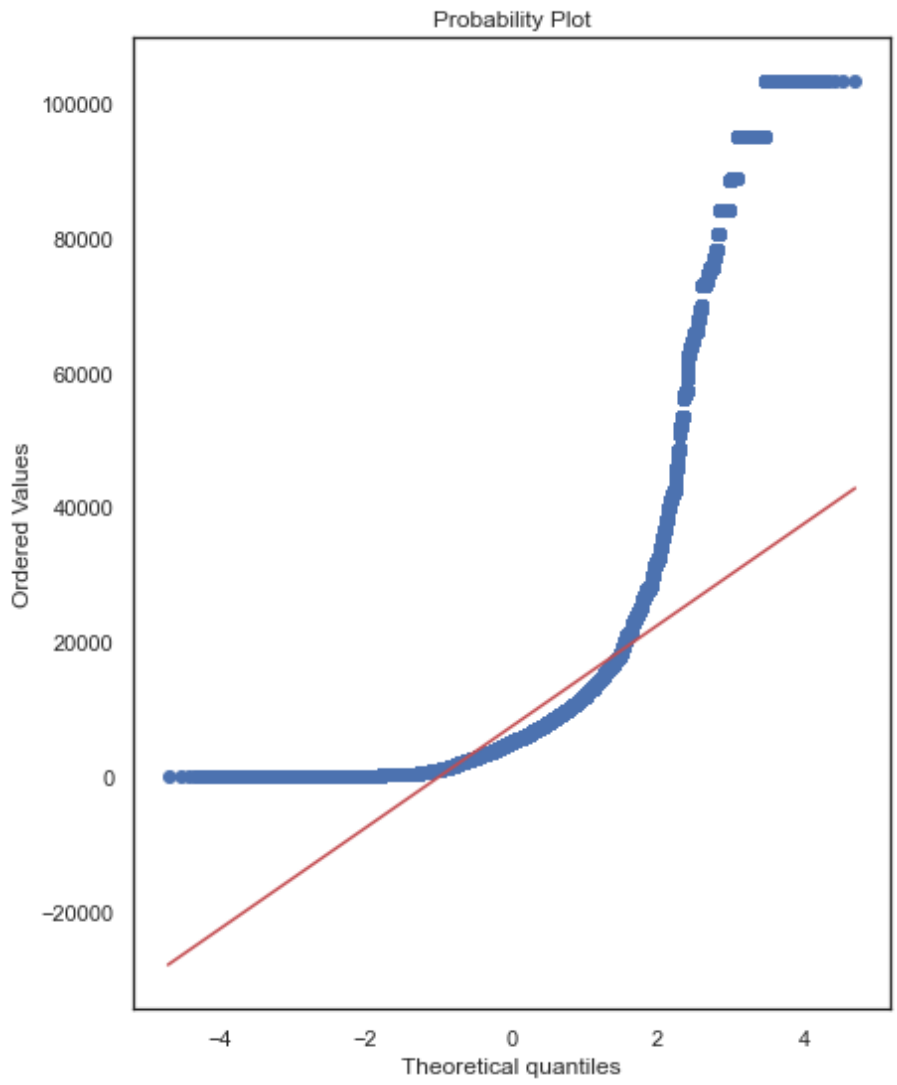
1. 'MarkDown1'
2. 'MarkDown2'
3. 'MarkDown3'
4. 'MarkDown4'
5. 'MarkDown5'
6. 'markdown_sum'
7. 'markdown_mean'
8. 'markdown_std'
9. 'markdown_median'
10. 'CPI'
11. 'Unemployment'

In [108]:
```
numeric_features_norm = ['MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4','MarkDown5','markdown_sum',
                         'markdown_mean', 'markdown_std', 'markdown_median', 'CPI', 'Unemployment']
```

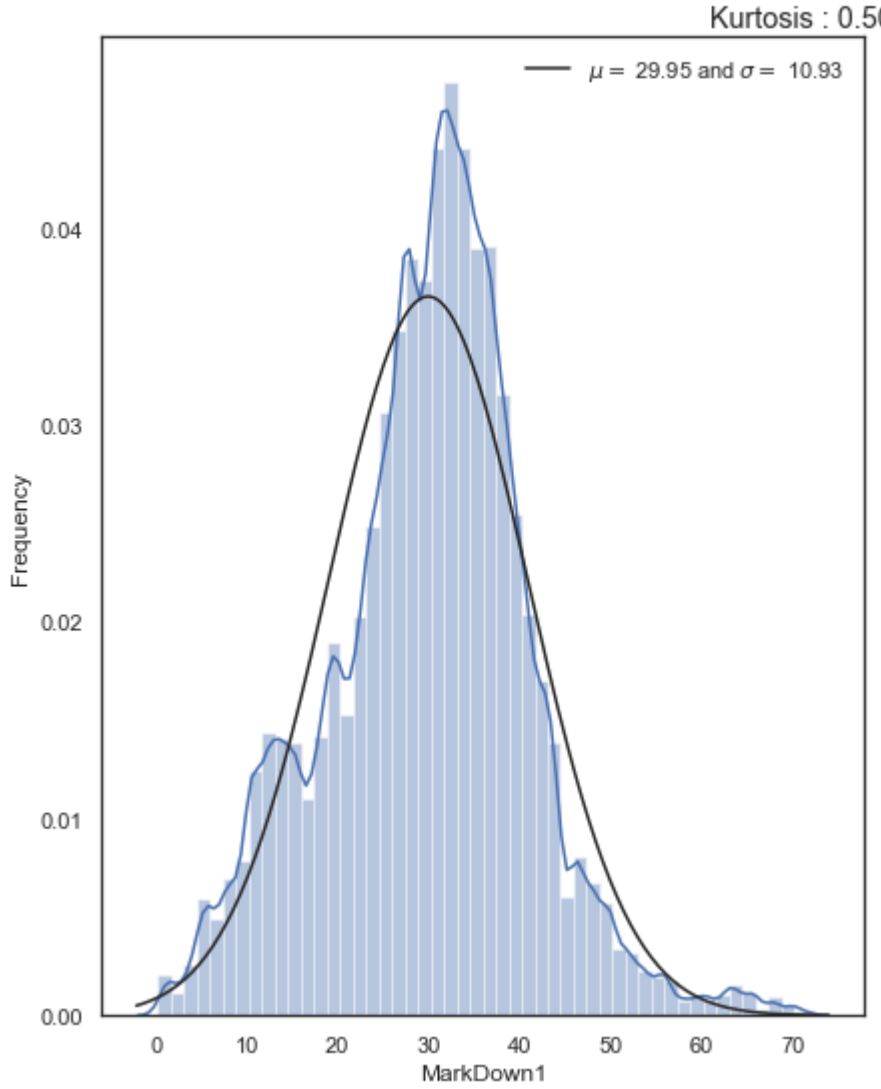
In [109]:
```
import scipy.stats as st
from scipy.stats import boxcox_normmax
from scipy.special import boxcox1p

for feature in numeric_features_norm:
    fig=plt.figure(figsize=(15,9))
    #pic1
    plt.subplot(1,2,1)
    sns.distplot(df[feature],fit=st.norm)
    (mu,sigma)=st.norm.fit(df[feature])
    plt.legend(['$\mu=$ {:.2f} and $\sigma=$ {:.2f}'.format(mu,sigma)],loc='best')
    plt.ylabel('Frequency')
    #pic2
    plt.subplot(1,2,2)
    res=st.probplot(df[feature],plot=plt)
    plt.suptitle(feature +'   ' + 'Before' + '\n'+f"Skewness : {df[feature].skew()}" + '\n'+f"Kurtosis : {df[f
eature].kurt()}")

    lambda_1=boxcox_normmax(df[feature]+1)
    df[feature]=boxcox1p(df[feature],lambda_1)

    fig=plt.figure(figsize=(15,9))
    #pic1
    plt.subplot(1,2,1)
    sns.distplot(df[feature],fit=st.norm)
    (mu,sigma)=st.norm.fit(df[feature])
    plt.legend(['$\mu=$ {:.2f} and $\sigma=$ {:.2f}'.format(mu,sigma)],loc='best')
    plt.ylabel('Frequency')
    #pic2
    plt.subplot(1,2,2)
    res=st.probplot(df[feature],plot=plt)
    plt.suptitle(feature + '   ' + 'After' + '\n' + 'lambda:' + str(lambda_1) + '\n' + f"Skewness : {df[featur
e].skew()}" + '\n' + f"Kurtosis : {df[feature].kurt()}",va='top')
```

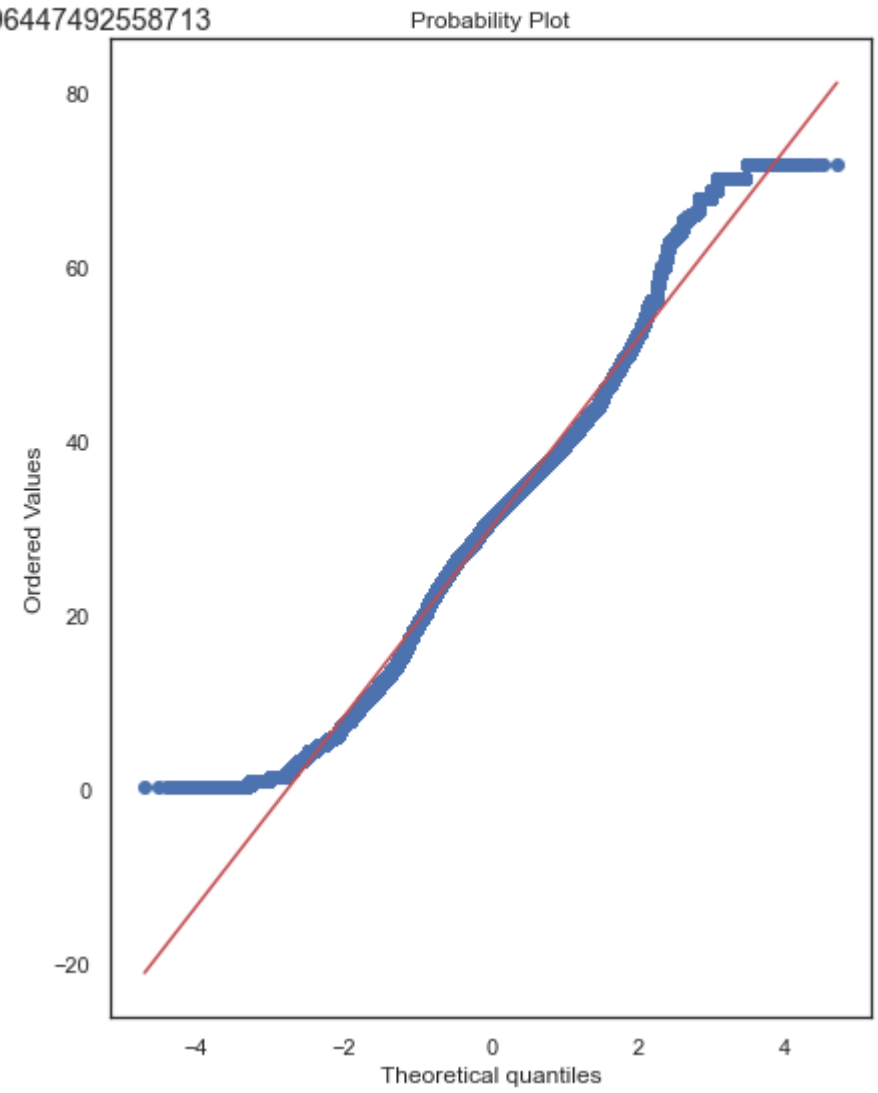
MarkDown1 Before
Skewness : 4.214876259020327
Kurtosis : 25.788174717622653



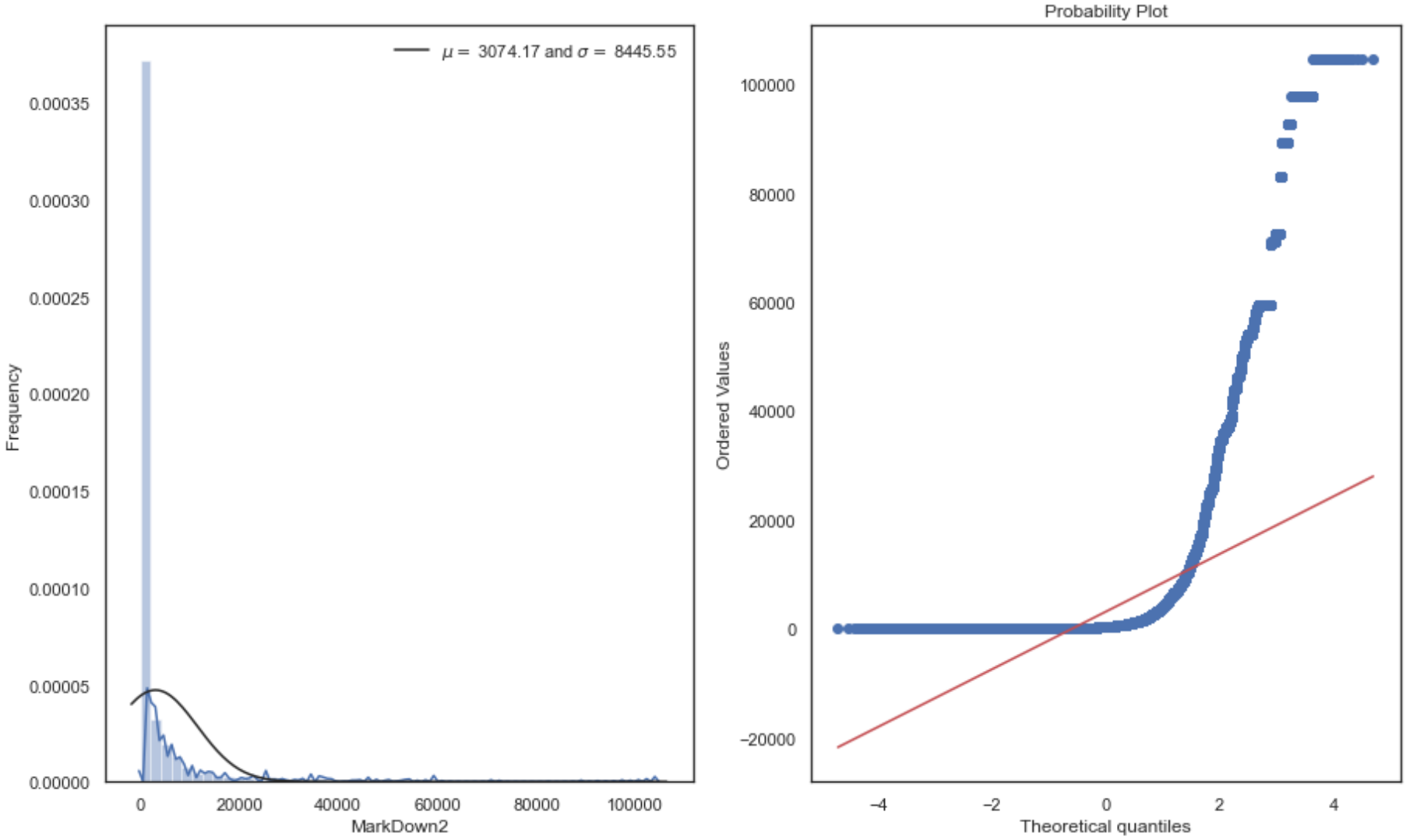


MarkDown1 After
lambda:0.2568493200404251
Skewness : 0.014639678938063434
Kurtosis : 0.5096447492558713

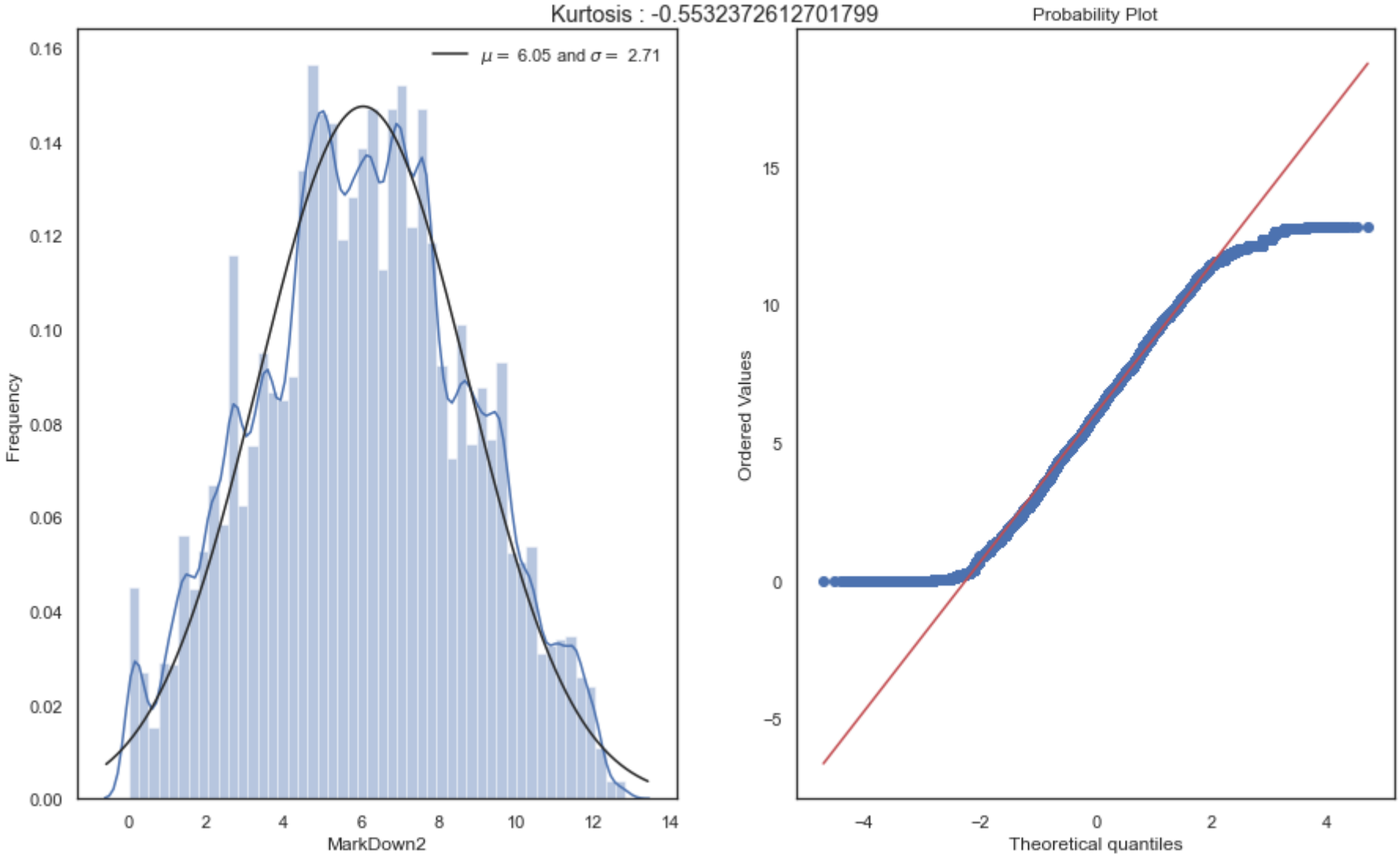




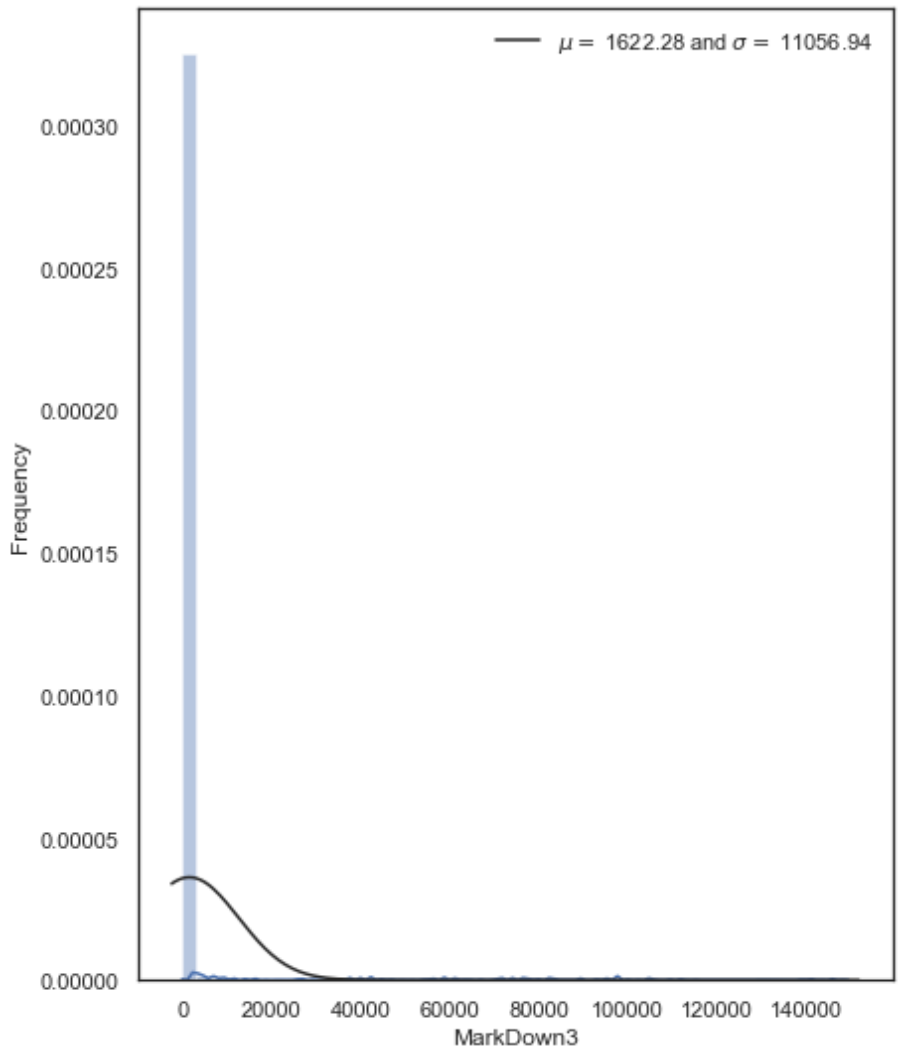
MarkDown2 Before
Skewness : 5.0360101347276816
Kurtosis : 32.08335435672573

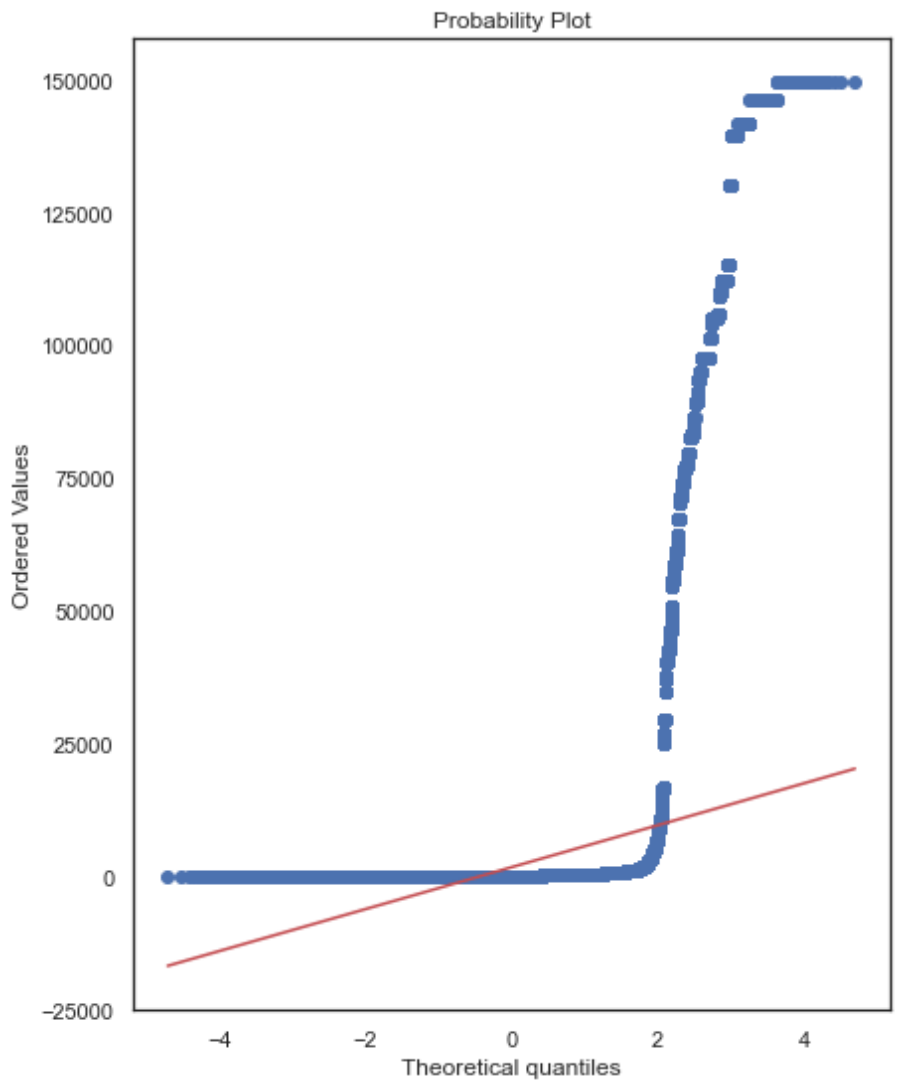


MarkDown2 After
lambda:0.017655472838515707
Skewness : 0.0005919145154430875
Kurtosis : -0.5532372612701799

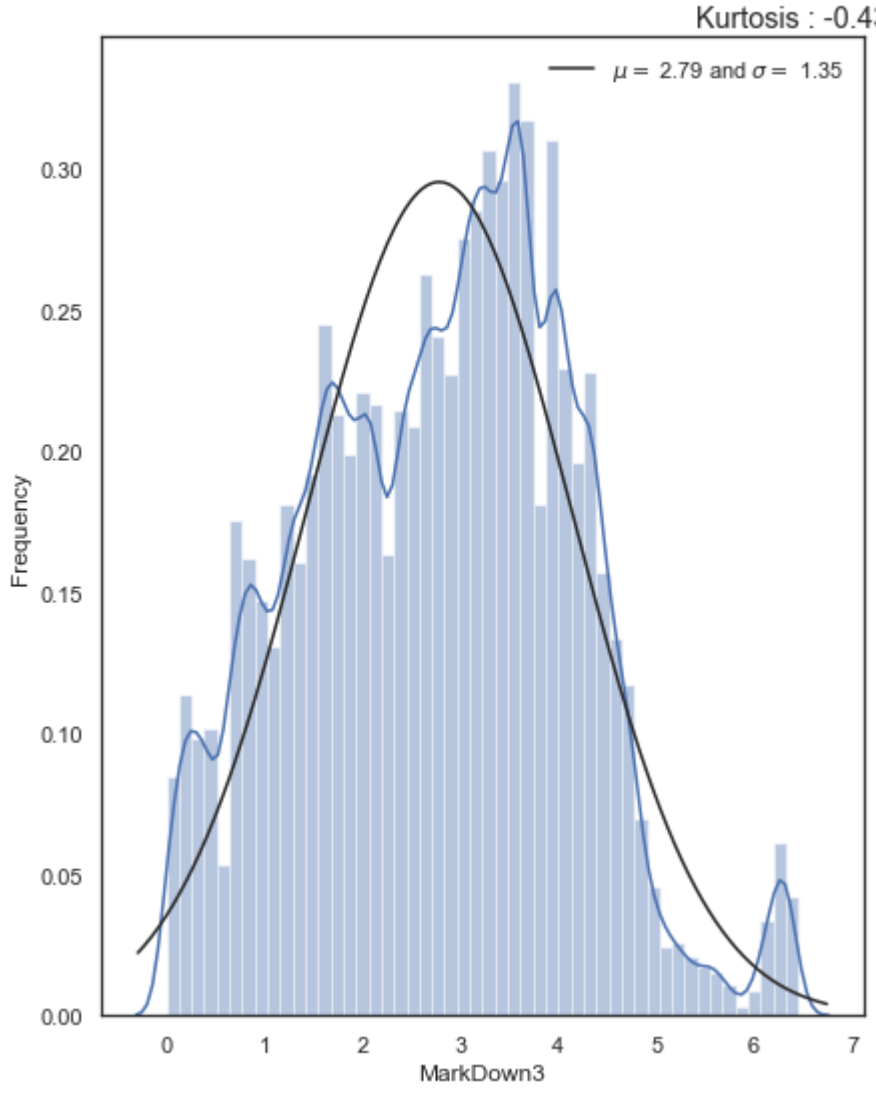


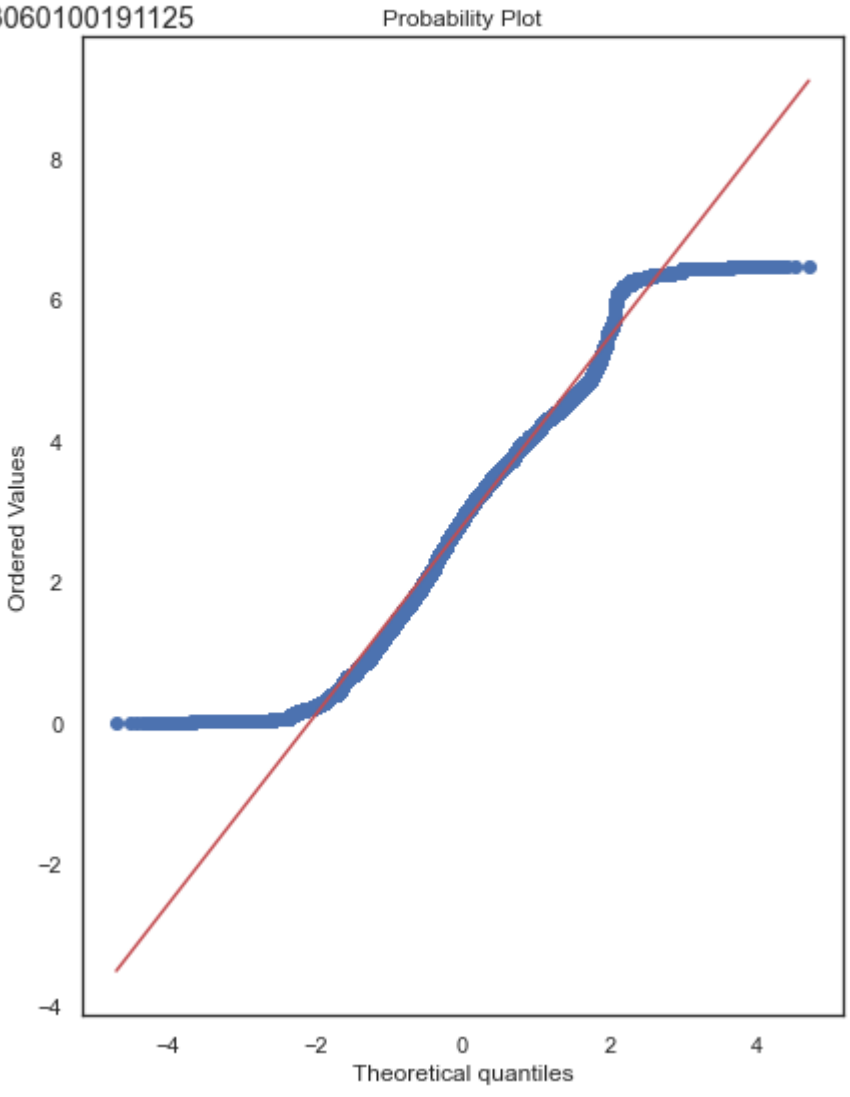
MarkDown3 Before
Skewness : 8.585765357735585
Kurtosis : 79.79381081246439



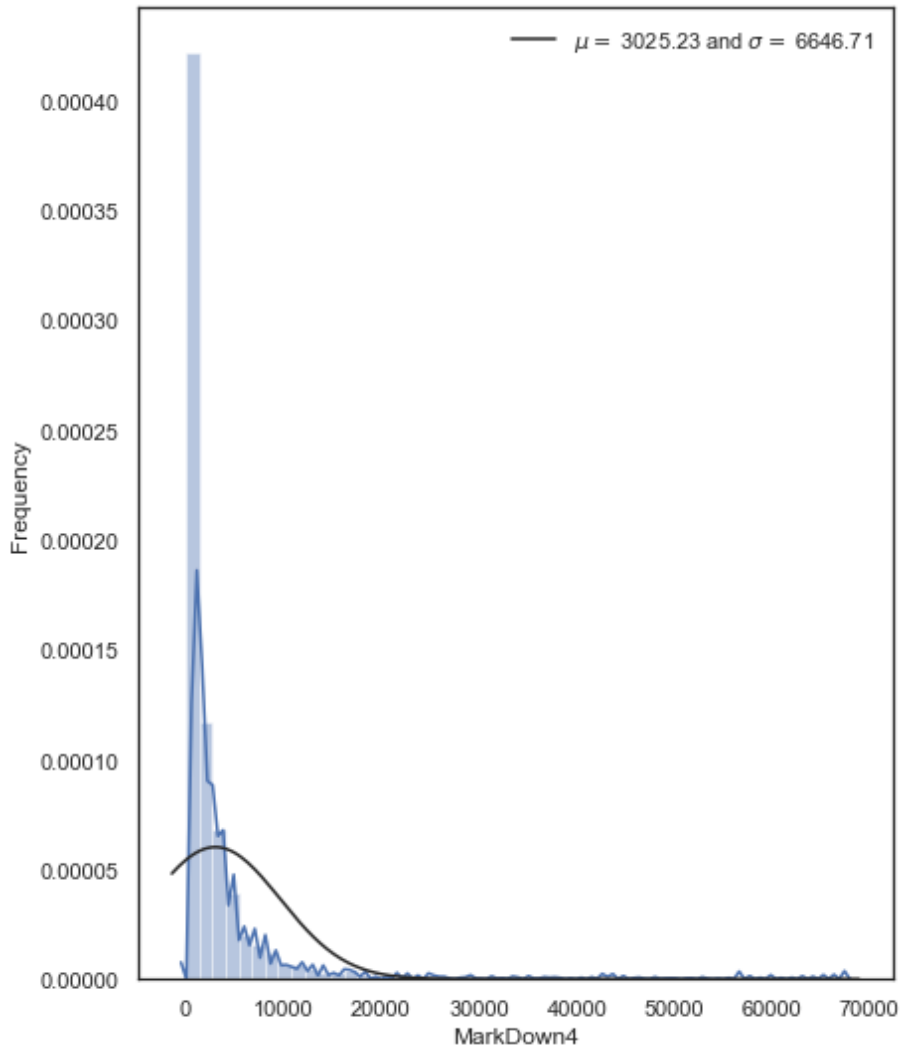


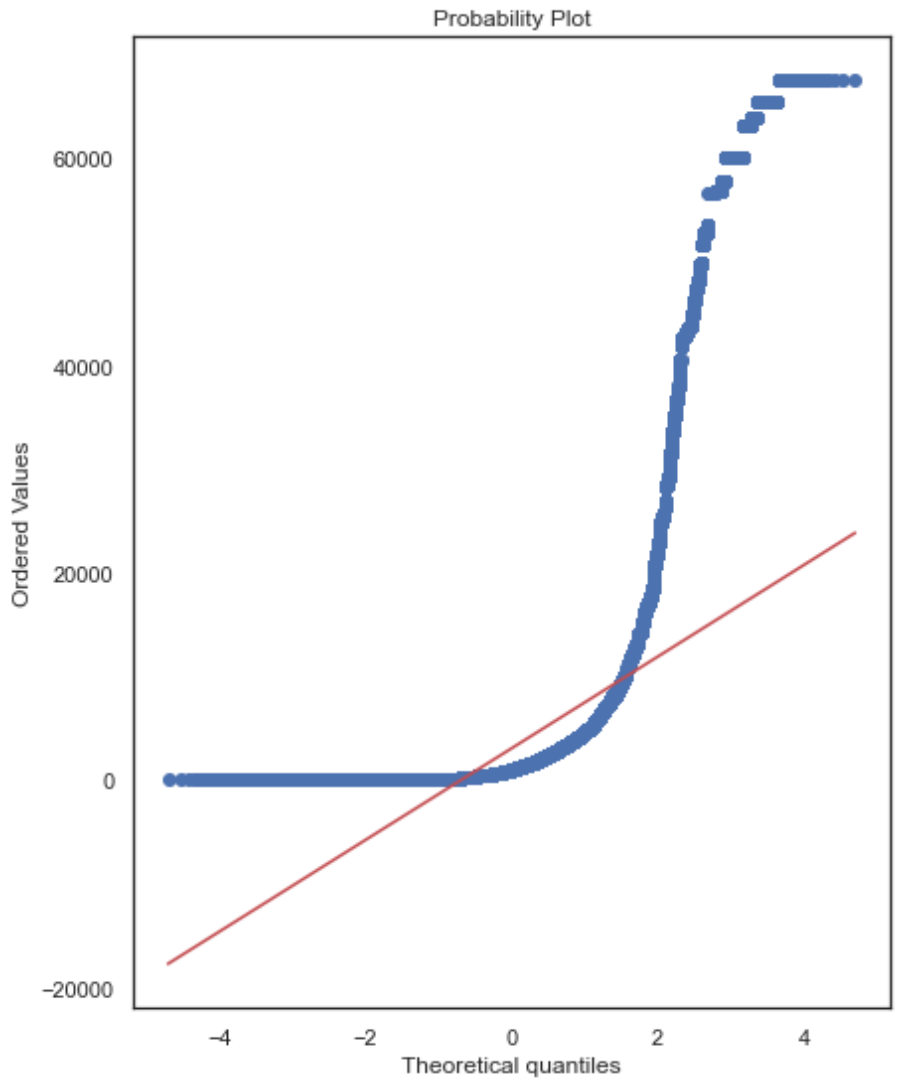
MarkDown3 After
lambda:-0.11608681660246019
Skewness : 0.03364537569201783
Kurtosis : -0.43213060100191125



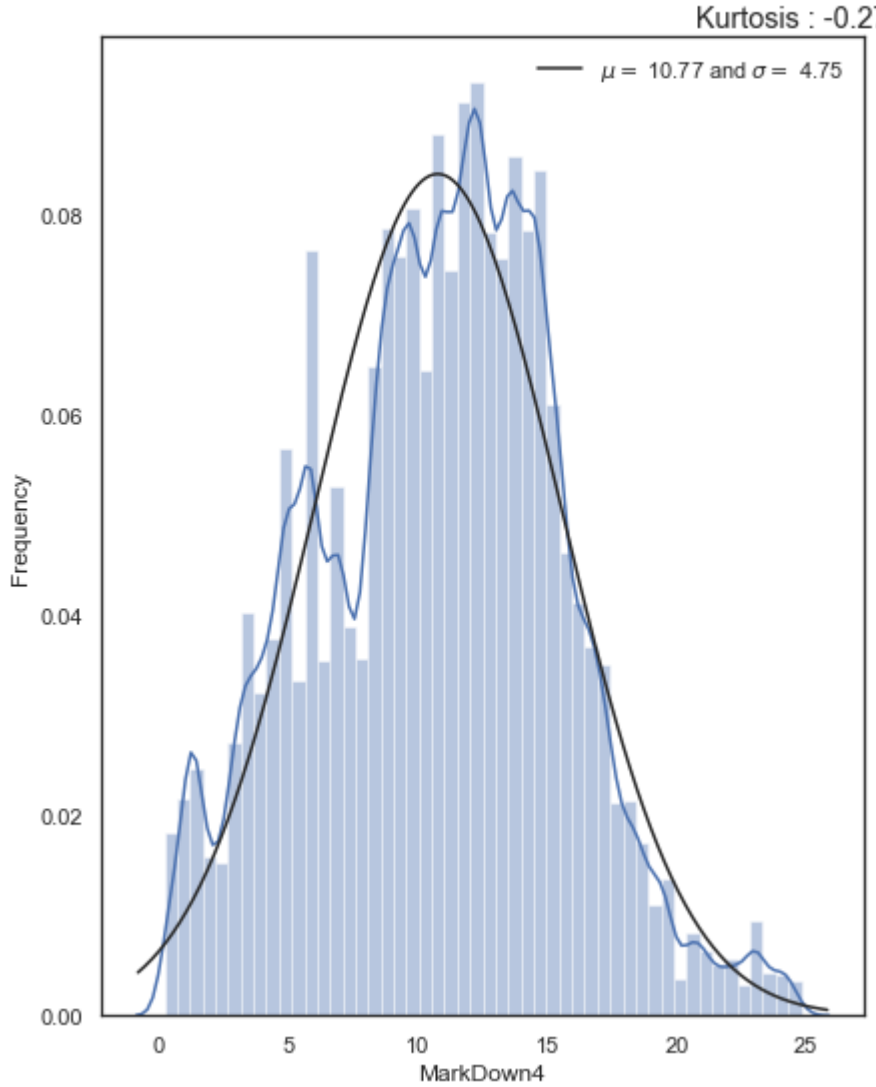


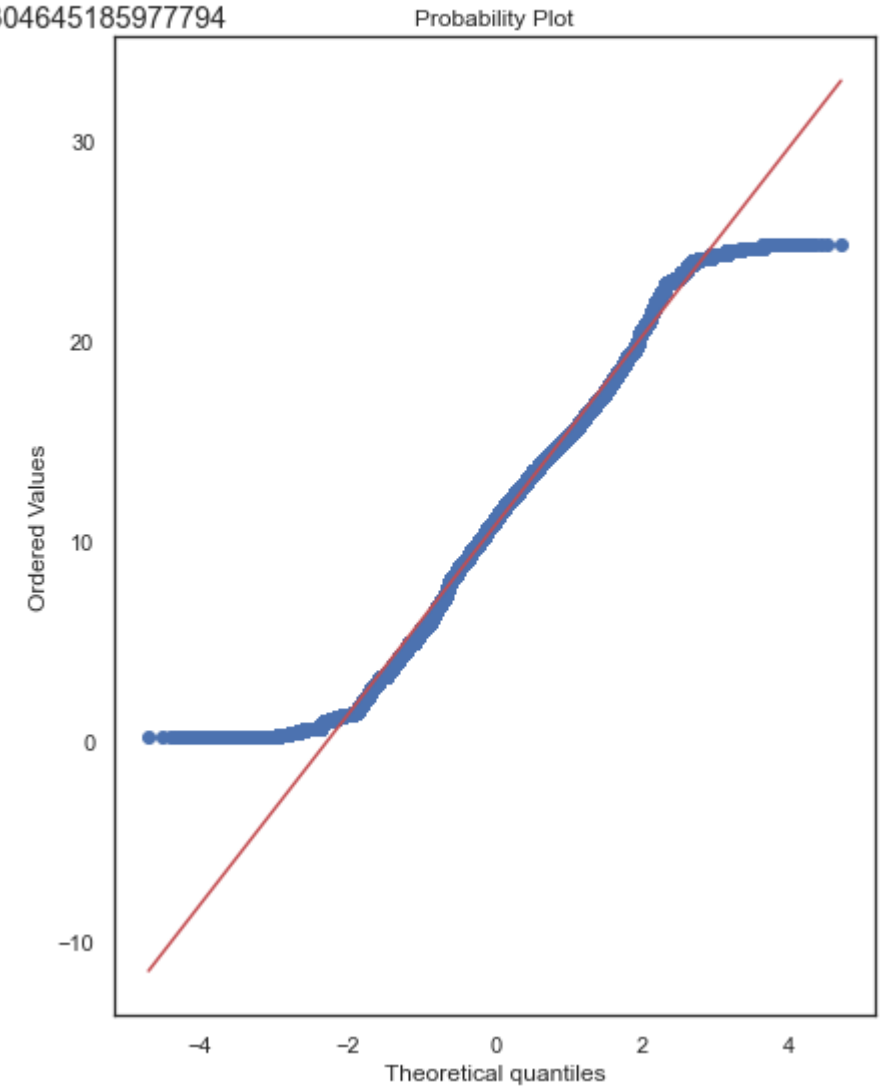
MarkDown4 Before
Skewness : 5.176850856780481
Kurtosis : 32.64372734448109



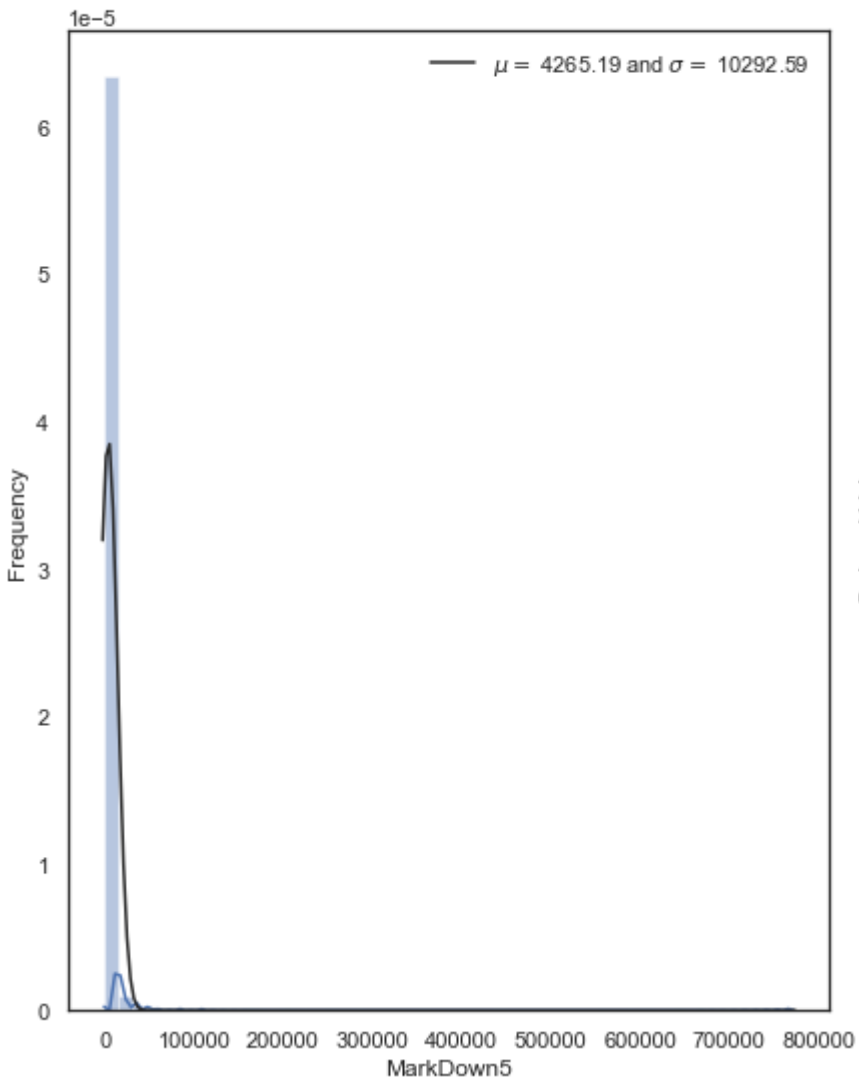


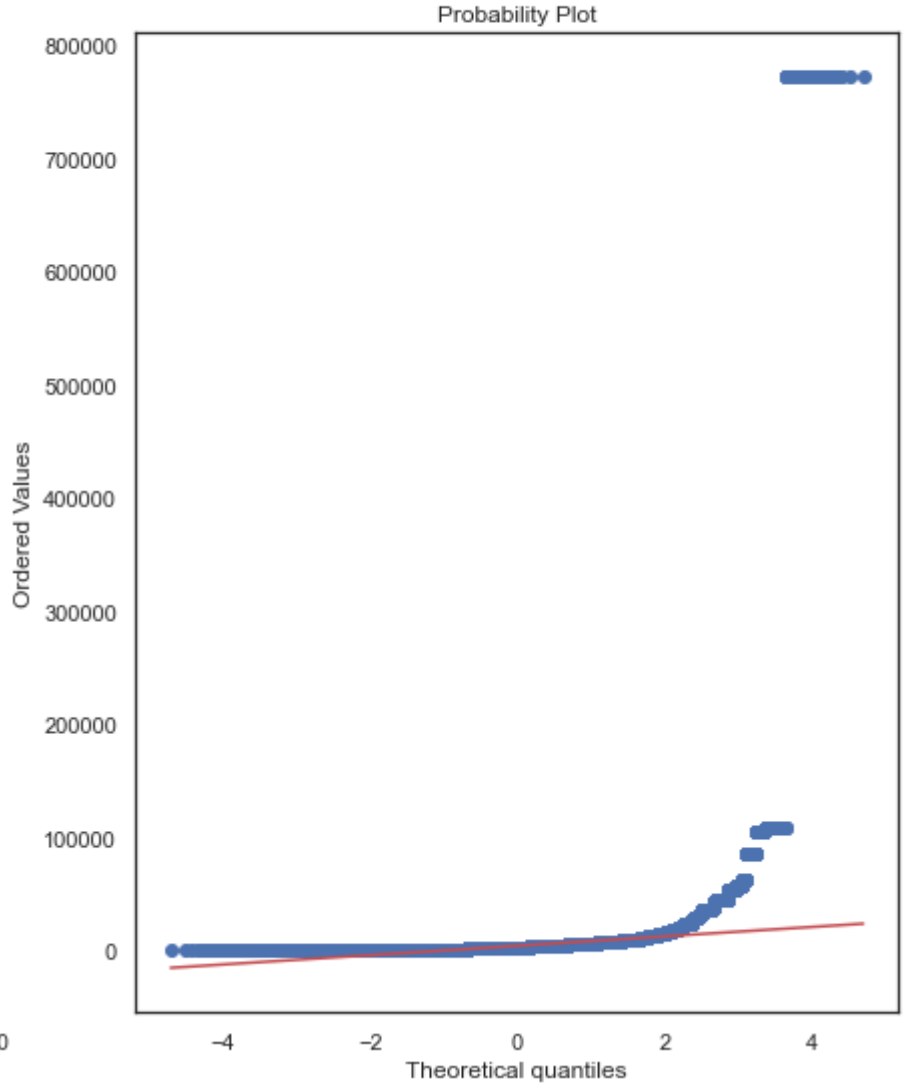
MarkDown4 After
lambda:0.12912677921012602
Skewness : 0.01124156121151023
Kurtosis : -0.27304645185977794



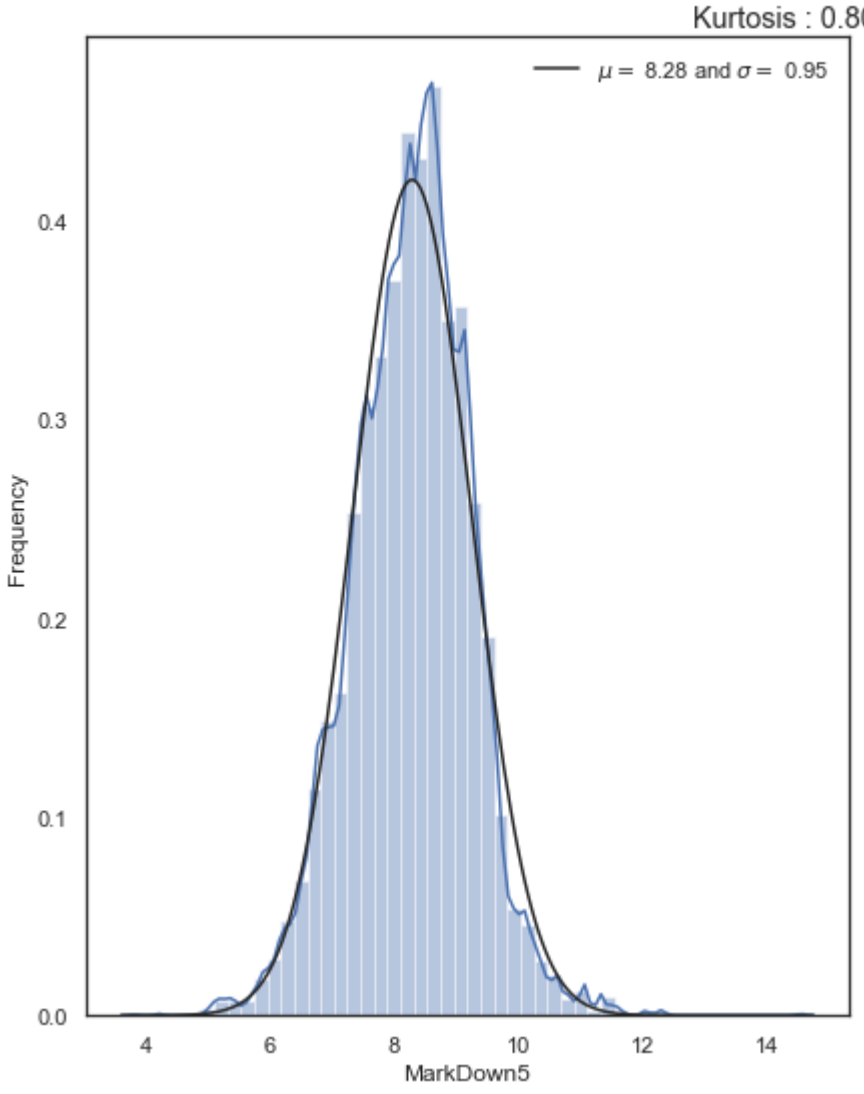


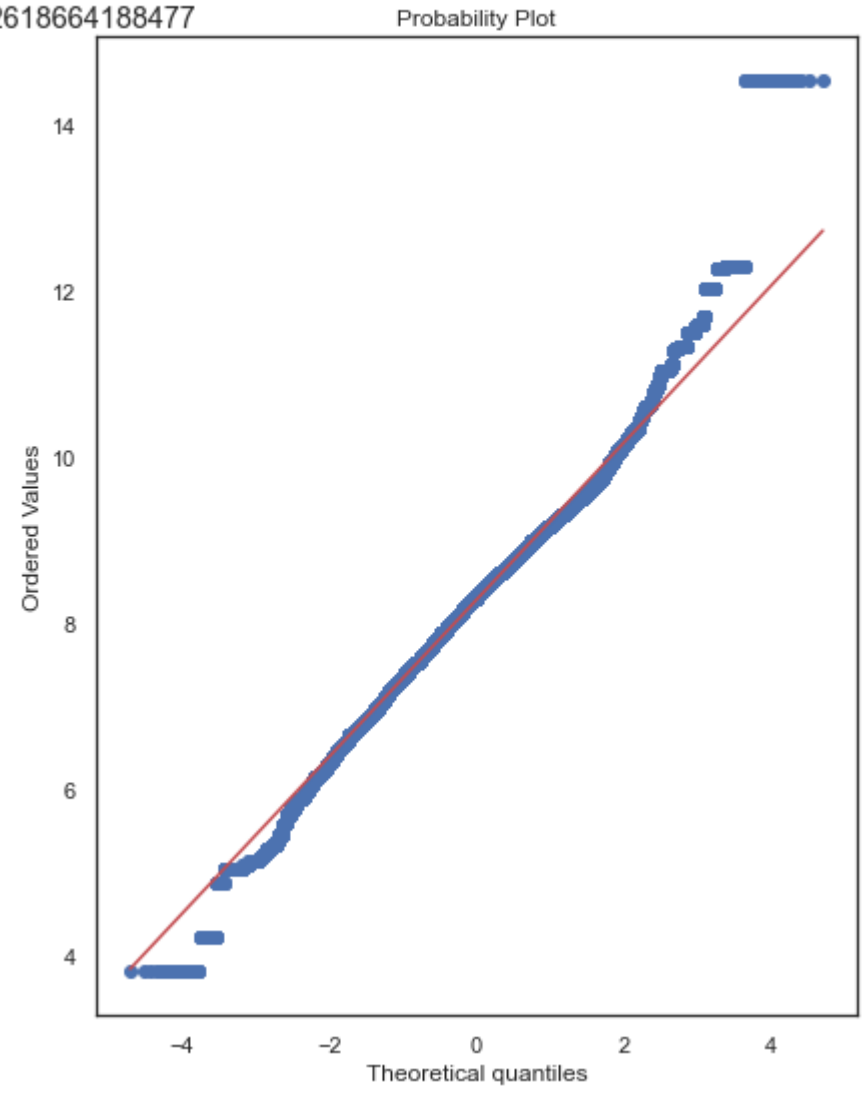
MarkDown5 Before
Skewness : 55.847094218728266
Kurtosis : 4088.489497803528



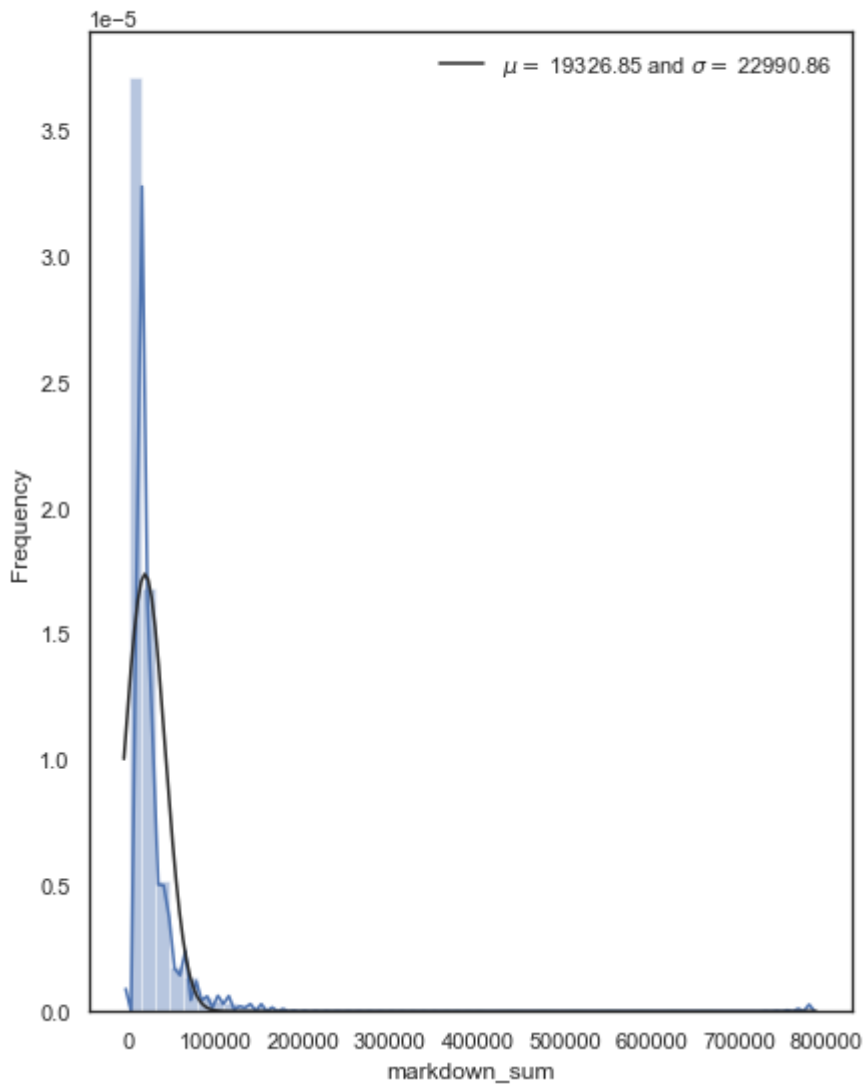


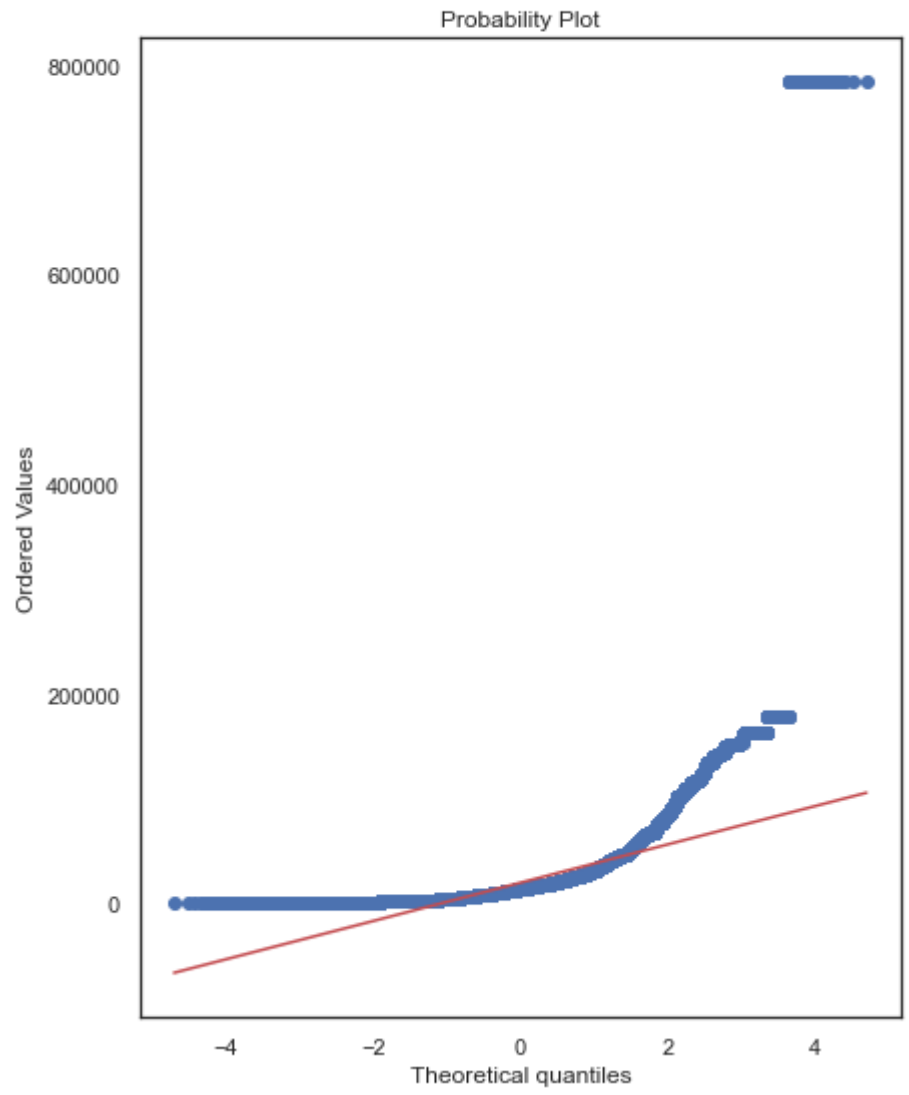
MarkDown5 After
lambda:0.010183996327251124
Skewness : -0.030632091714378685
Kurtosis : 0.8082618664188477



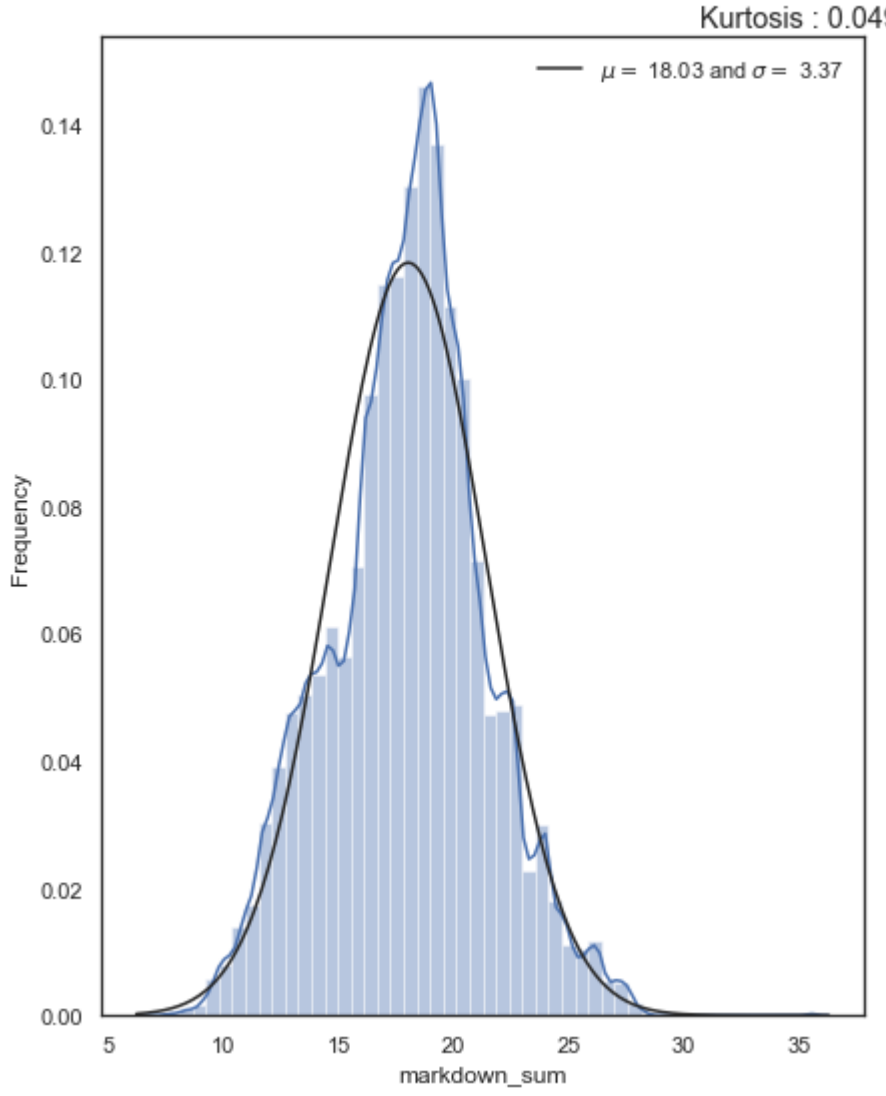


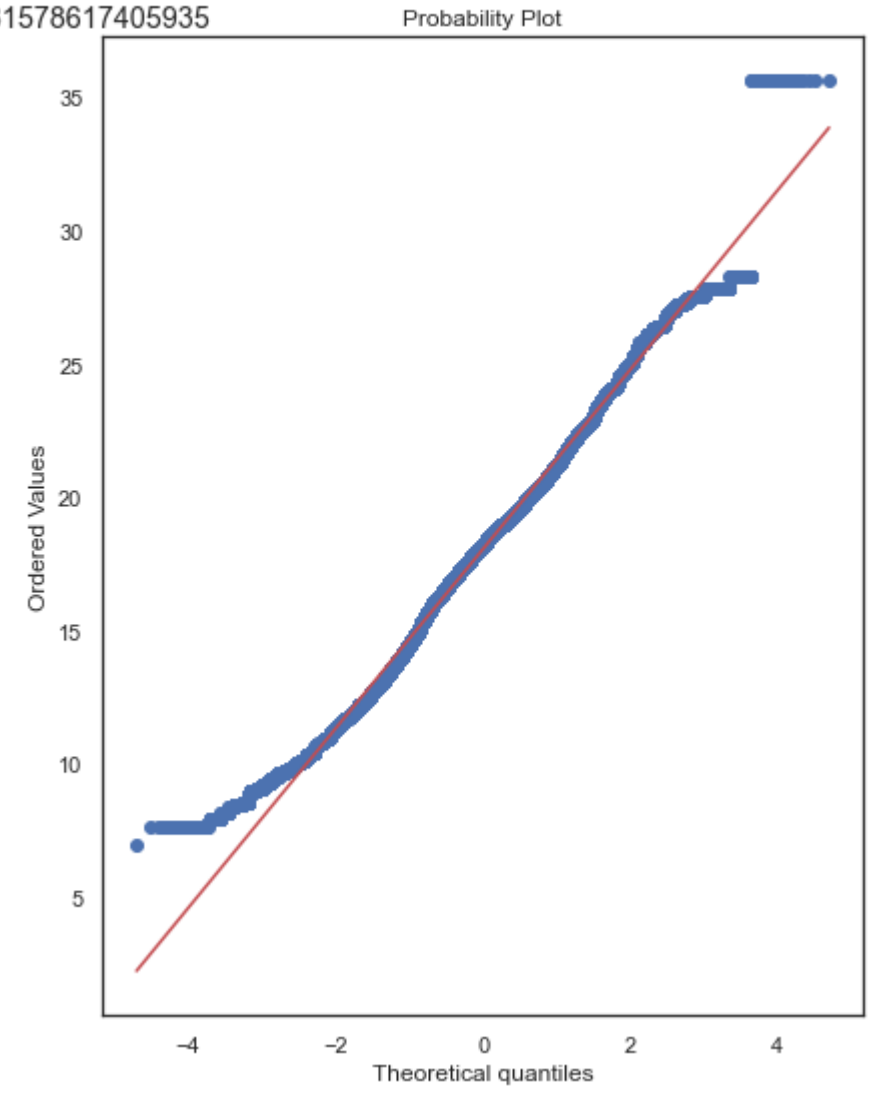
markdown_sum Before
Skewness : 7.11823769670529
Kurtosis : 168.74859713330073



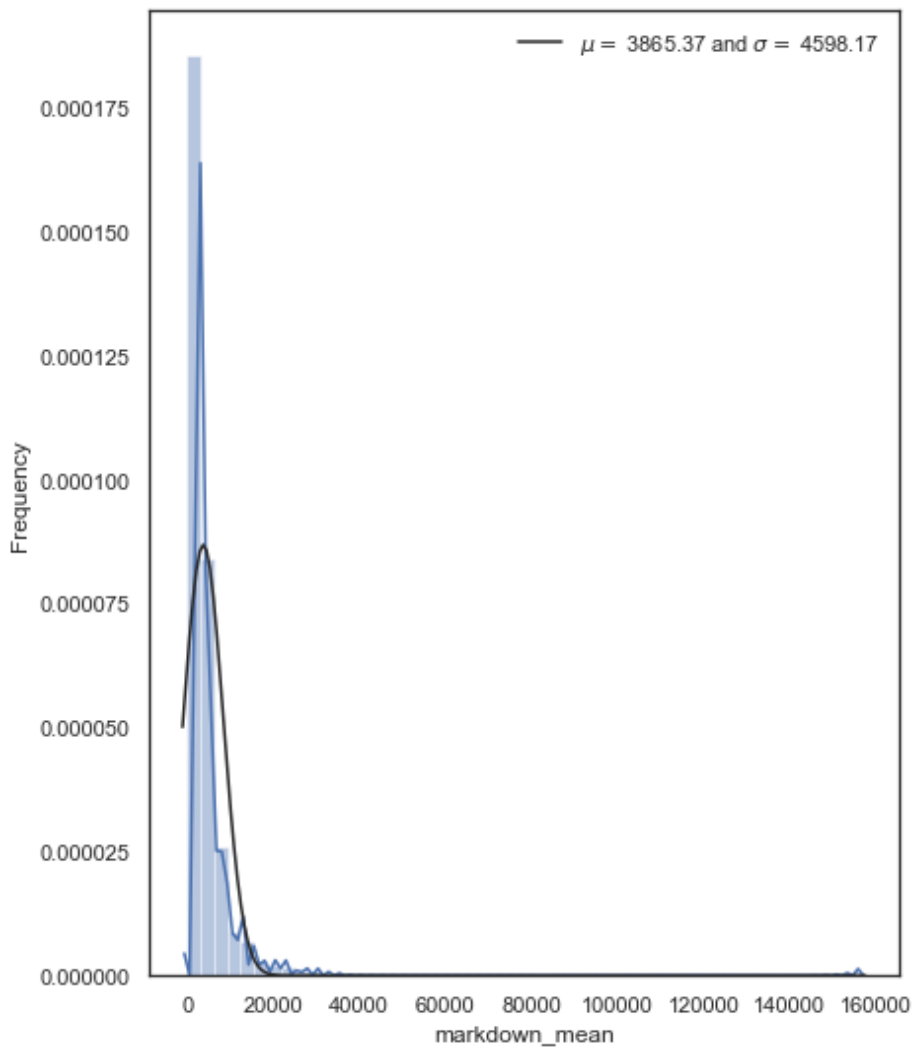


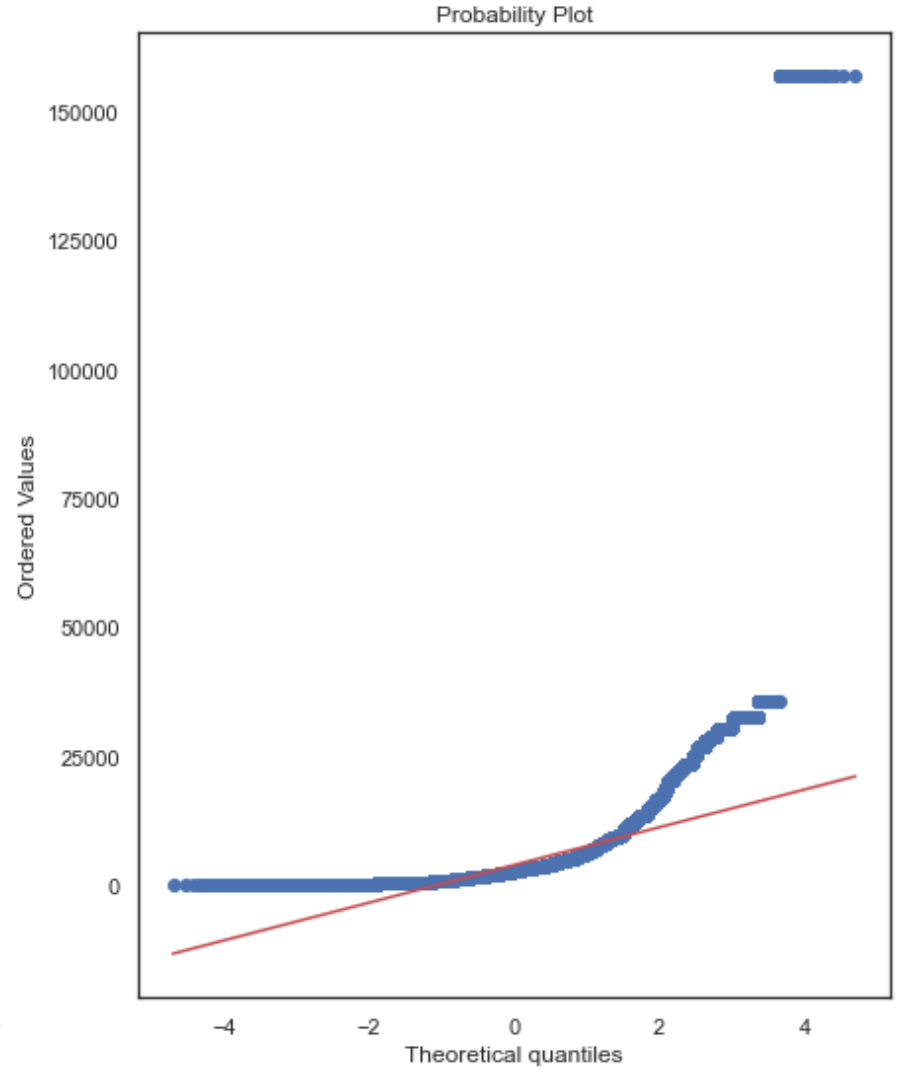
markdown_sum After
lambda:0.12491559510872124
Skewness : 0.0046316669528724845
Kurtosis : 0.04981578617405935



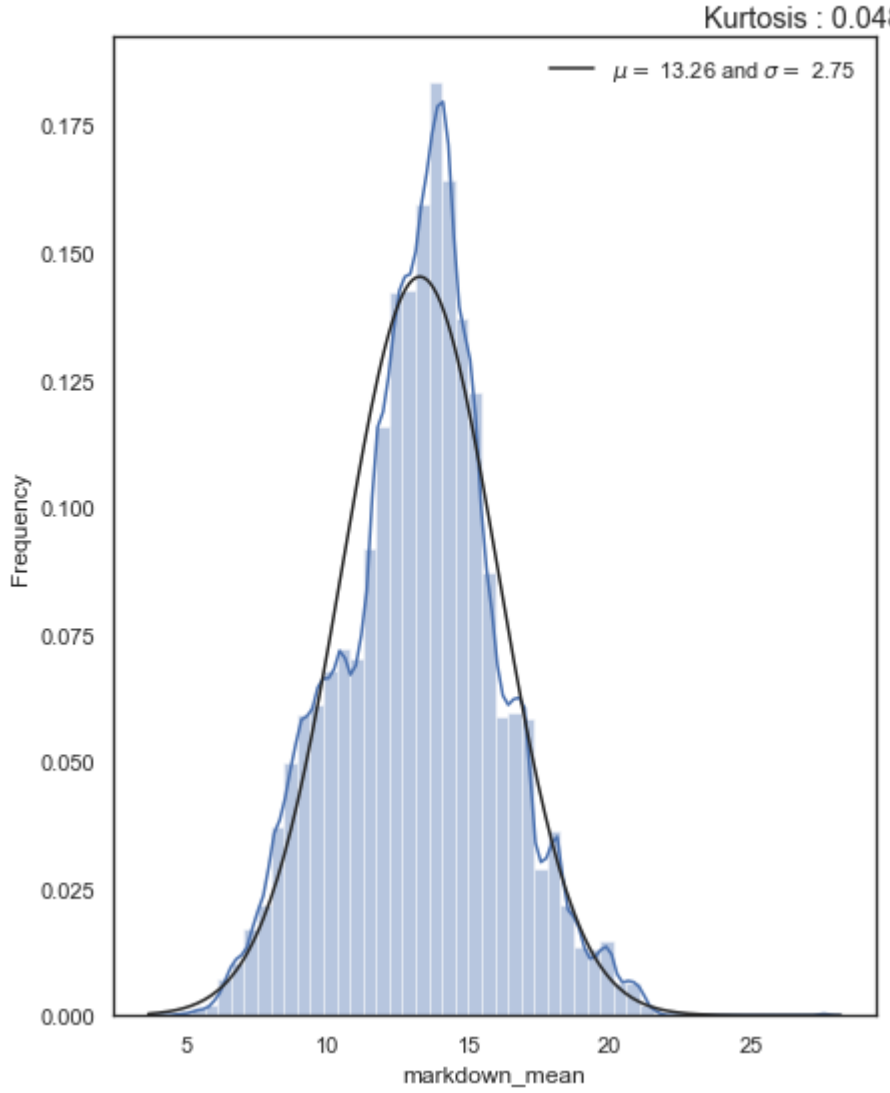


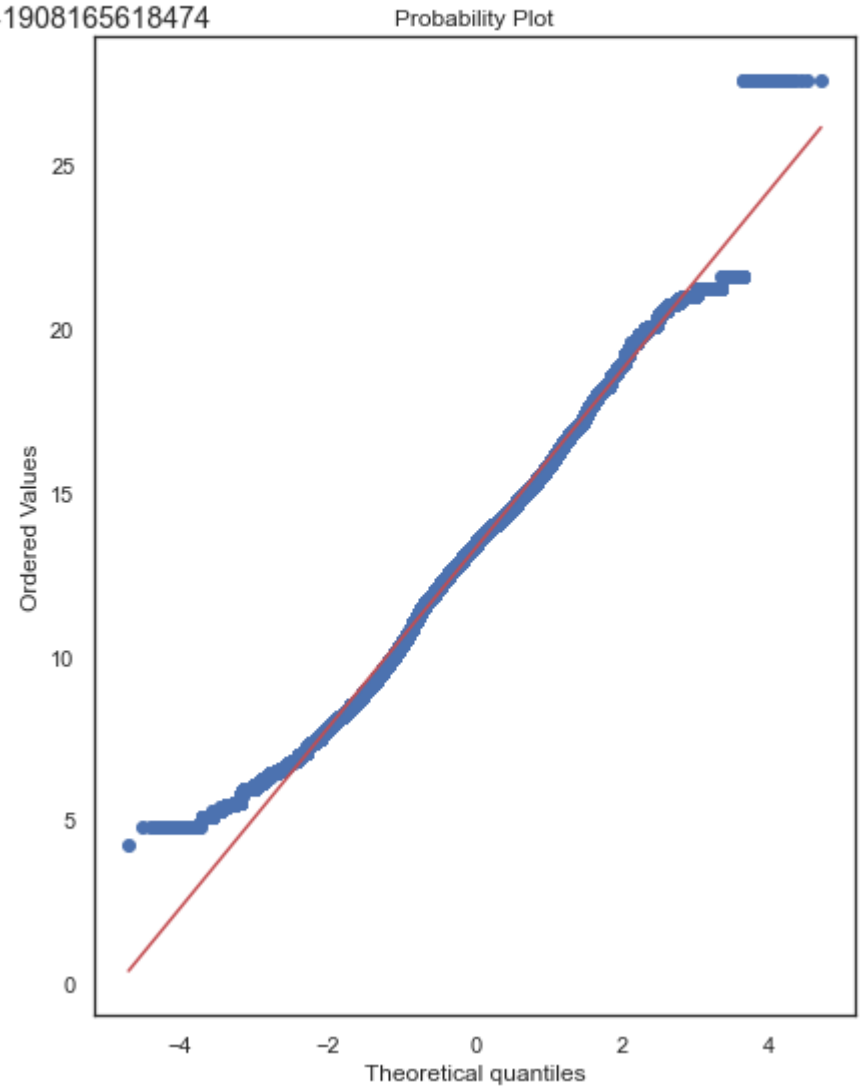
markdown_mean Before
Skewness : 7.118237696705279
Kurtosis : 168.7485971333006

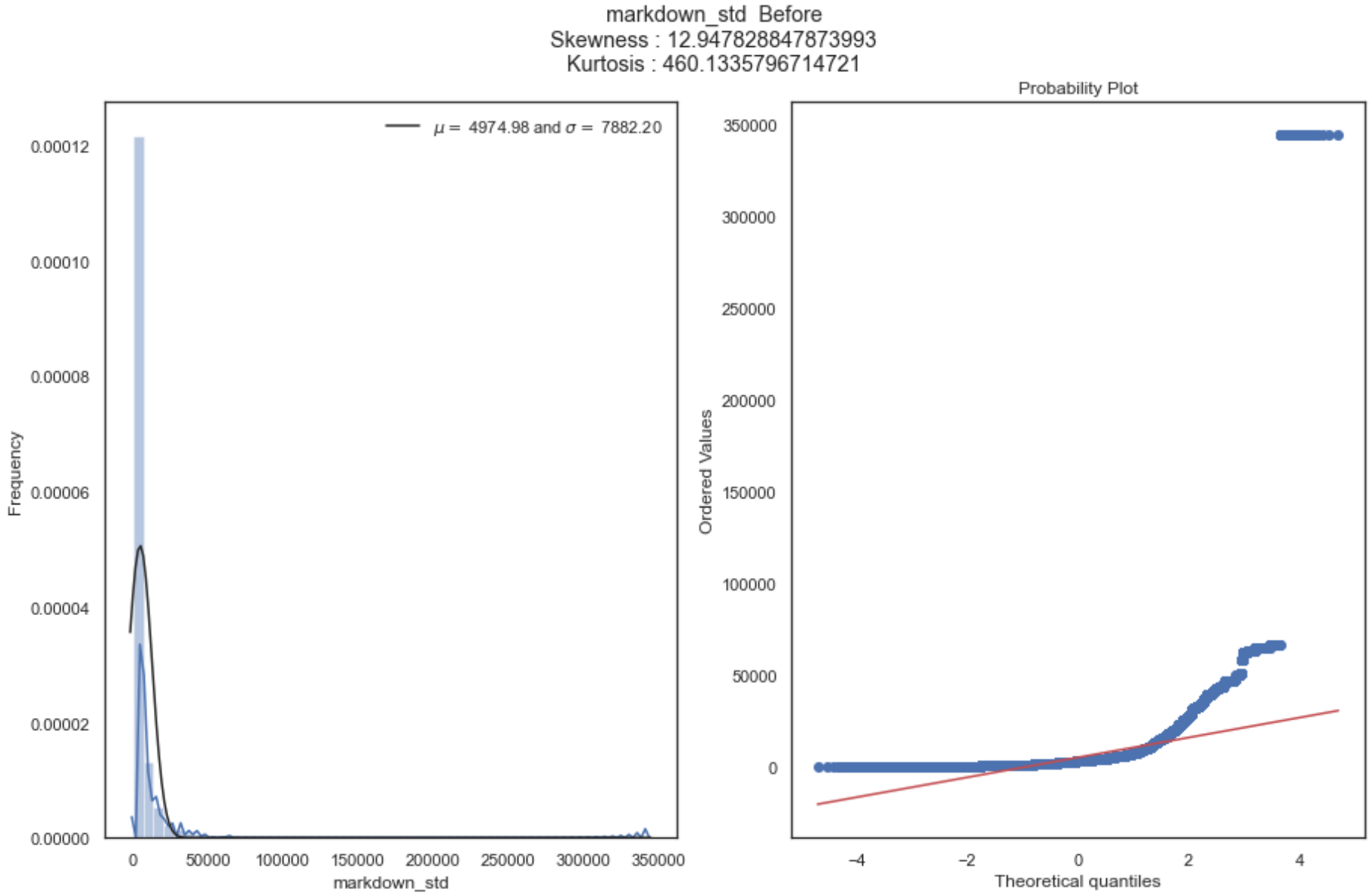


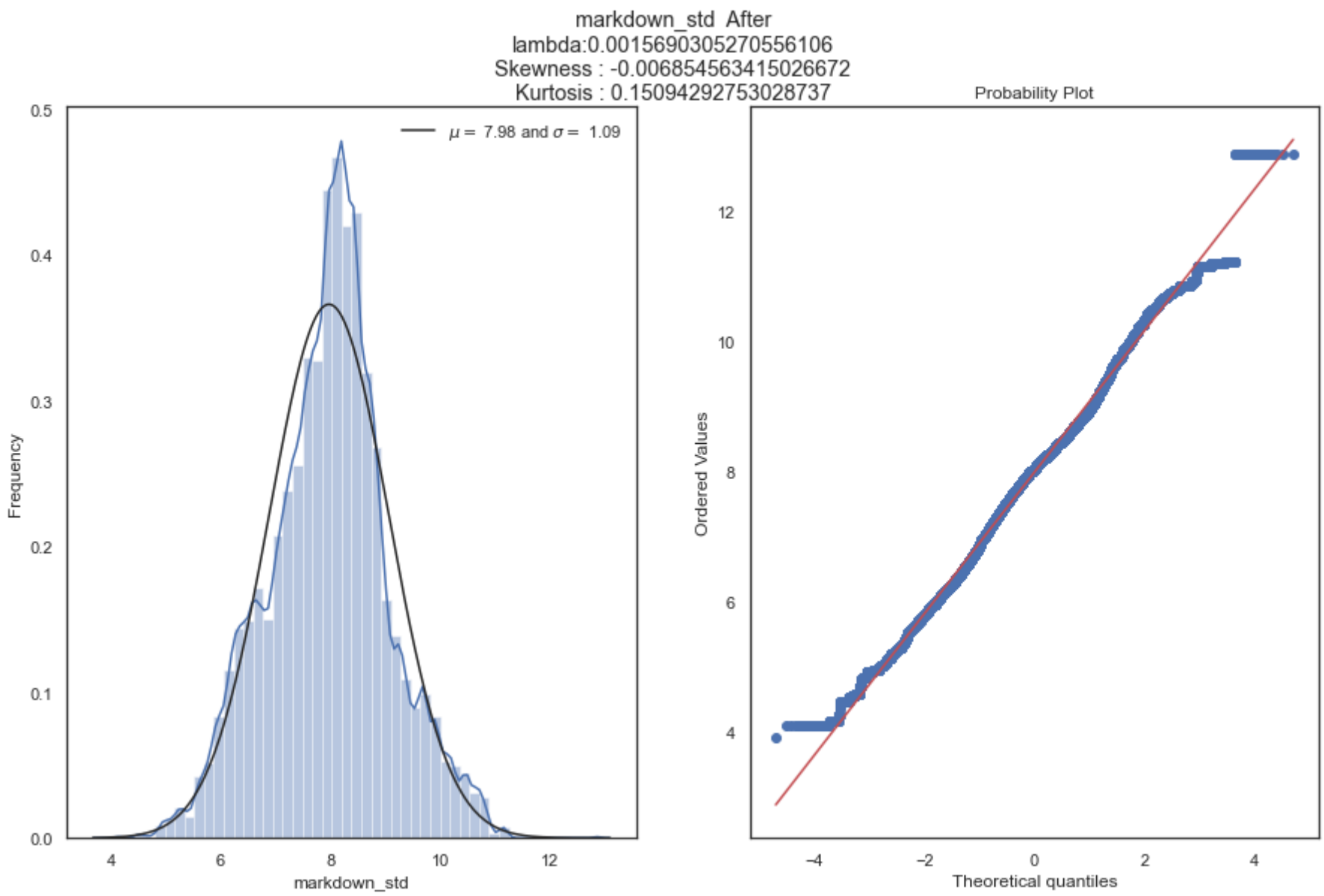


markdown_mean After
lambda:0.12445650236338619
Skewness : 0.004705239420465082
Kurtosis : 0.048141908165618474

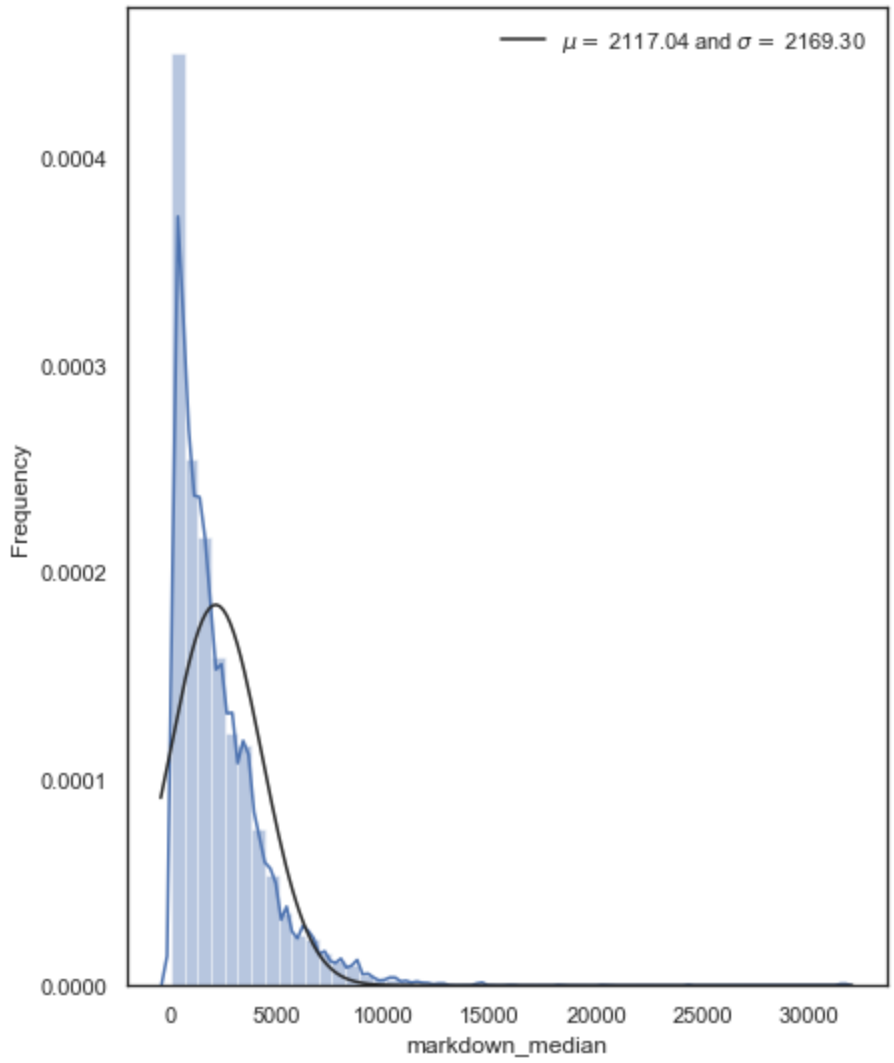


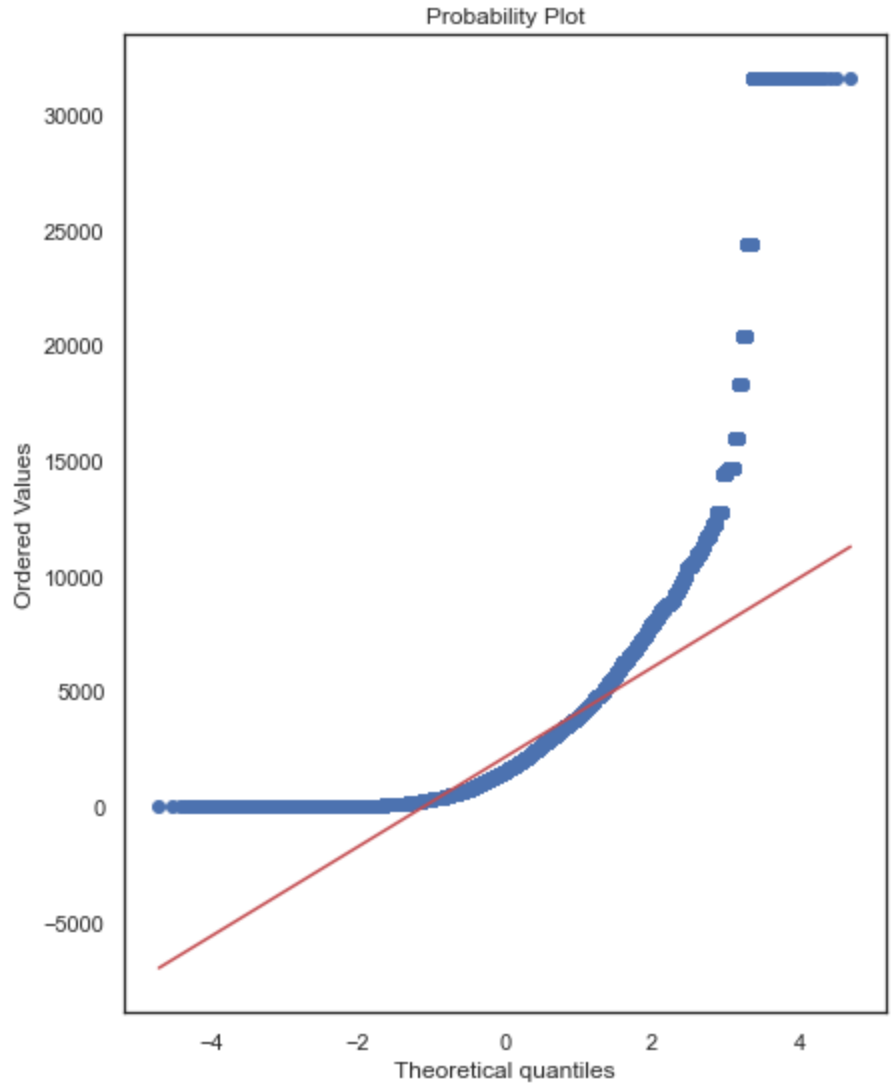



markdown_std Before
Skewness : 12.947828847873993
Kurtosis : 460.1335796714721
μ = 4974.98 and σ = 7882.20
Frequency
markdown_std
Probability Plot
Ordered Values
Theoretical quantiles

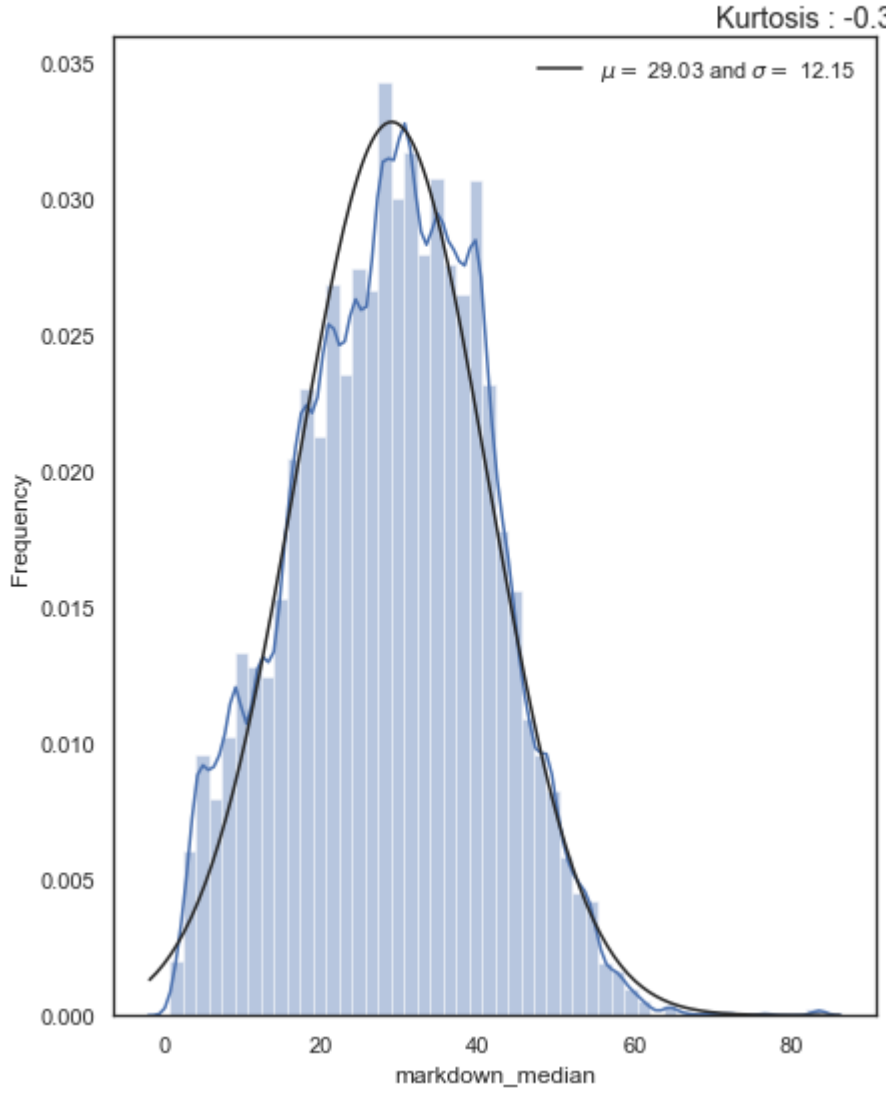

markdown_std After
lambda:0.0015690305270556106
Skewness : -0.006854563415026672
Kurtosis : 0.15094292753028737
μ = 7.98 and σ = 1.09
Frequency
markdown_std
Probability Plot
Ordered Values
Theoretical quantiles

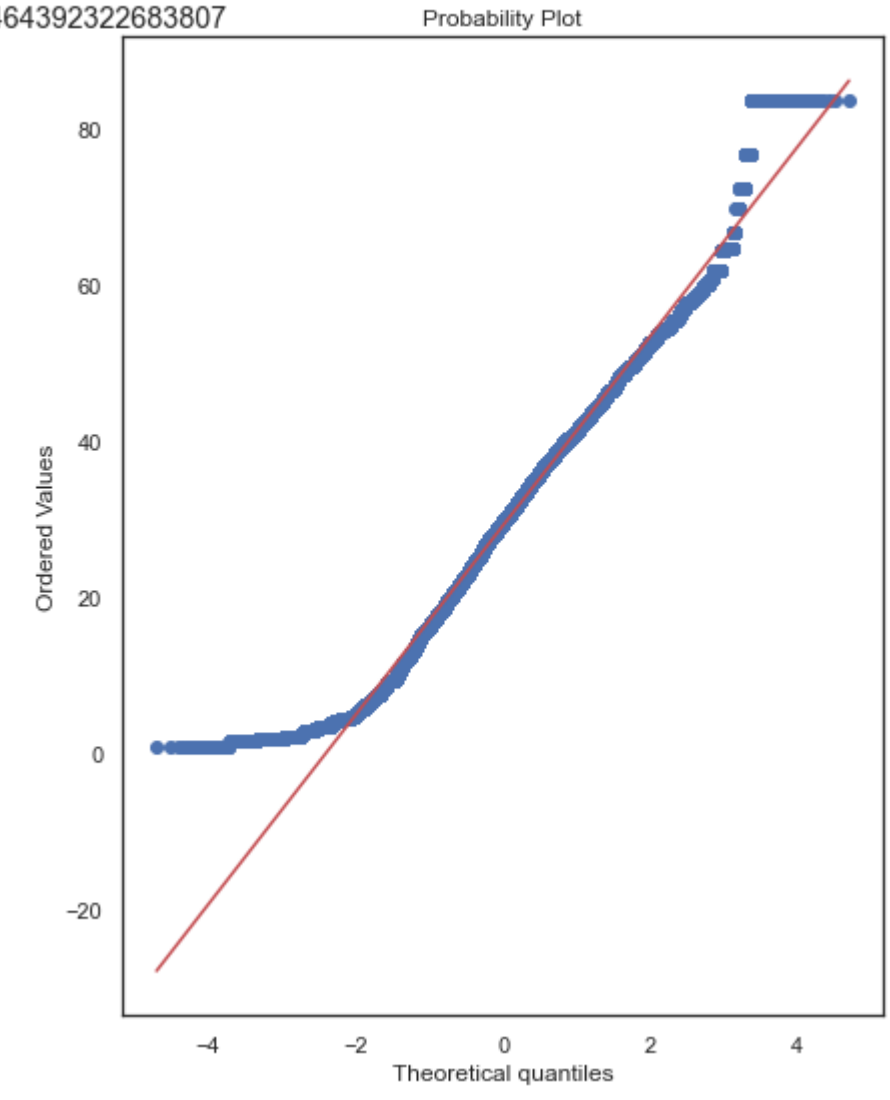
markdown_median Before
Skewness : 2.6278035016528465
Kurtosis : 17.90109152516868



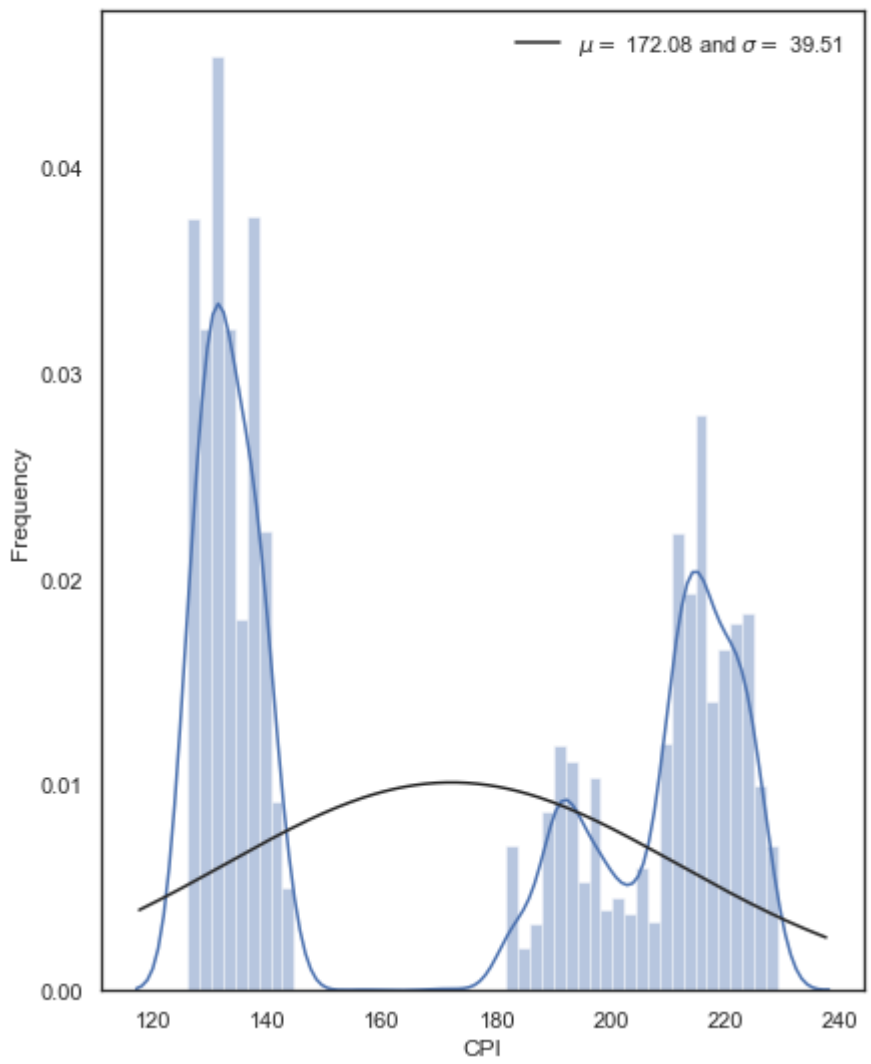


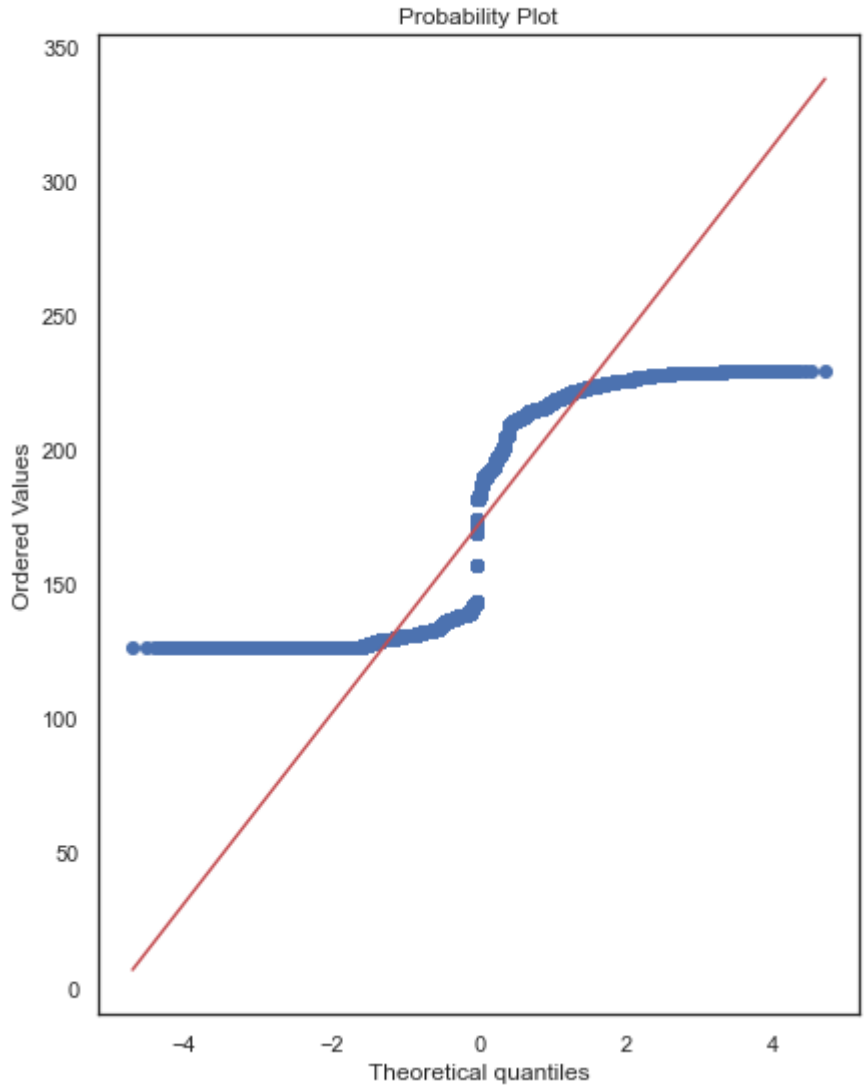
markdown_median After
lambda:0.32093319982527574
Skewness : -0.013333115007054384
Kurtosis : -0.3464392322683807



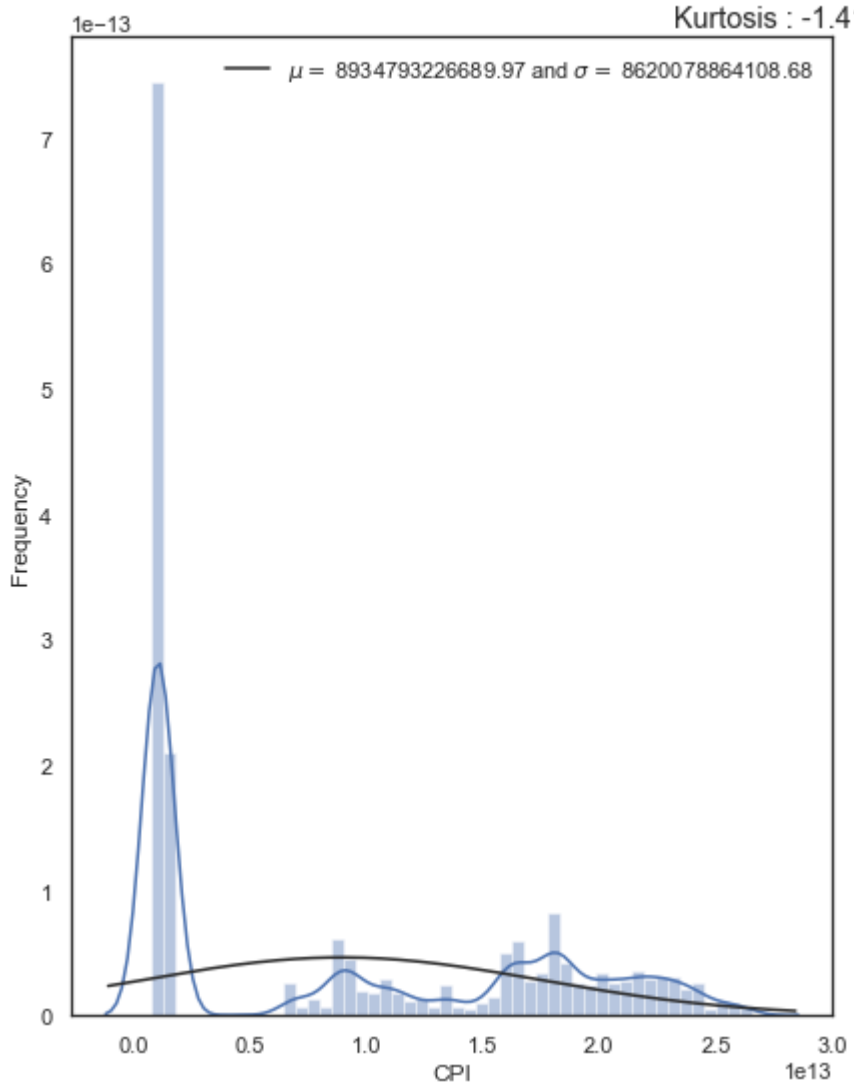


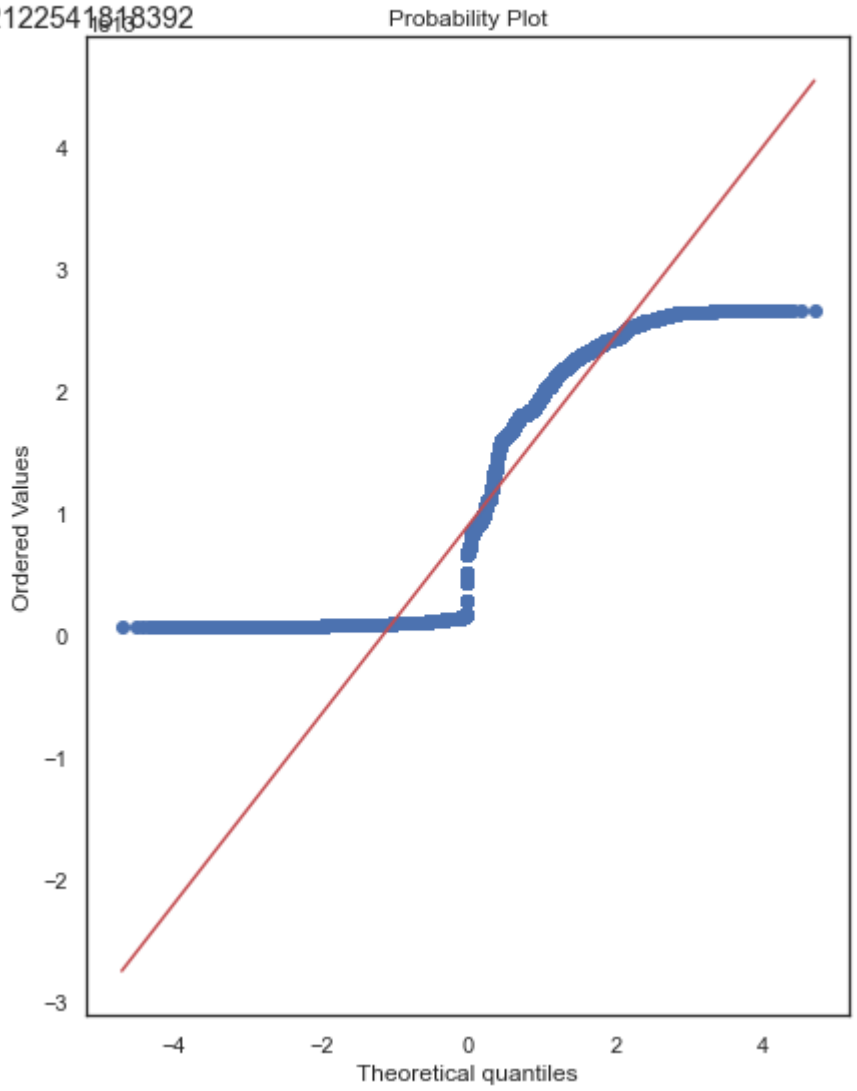
CPI Before
Skewness : 0.08713015114518347
Kurtosis : -1.8225702811366666

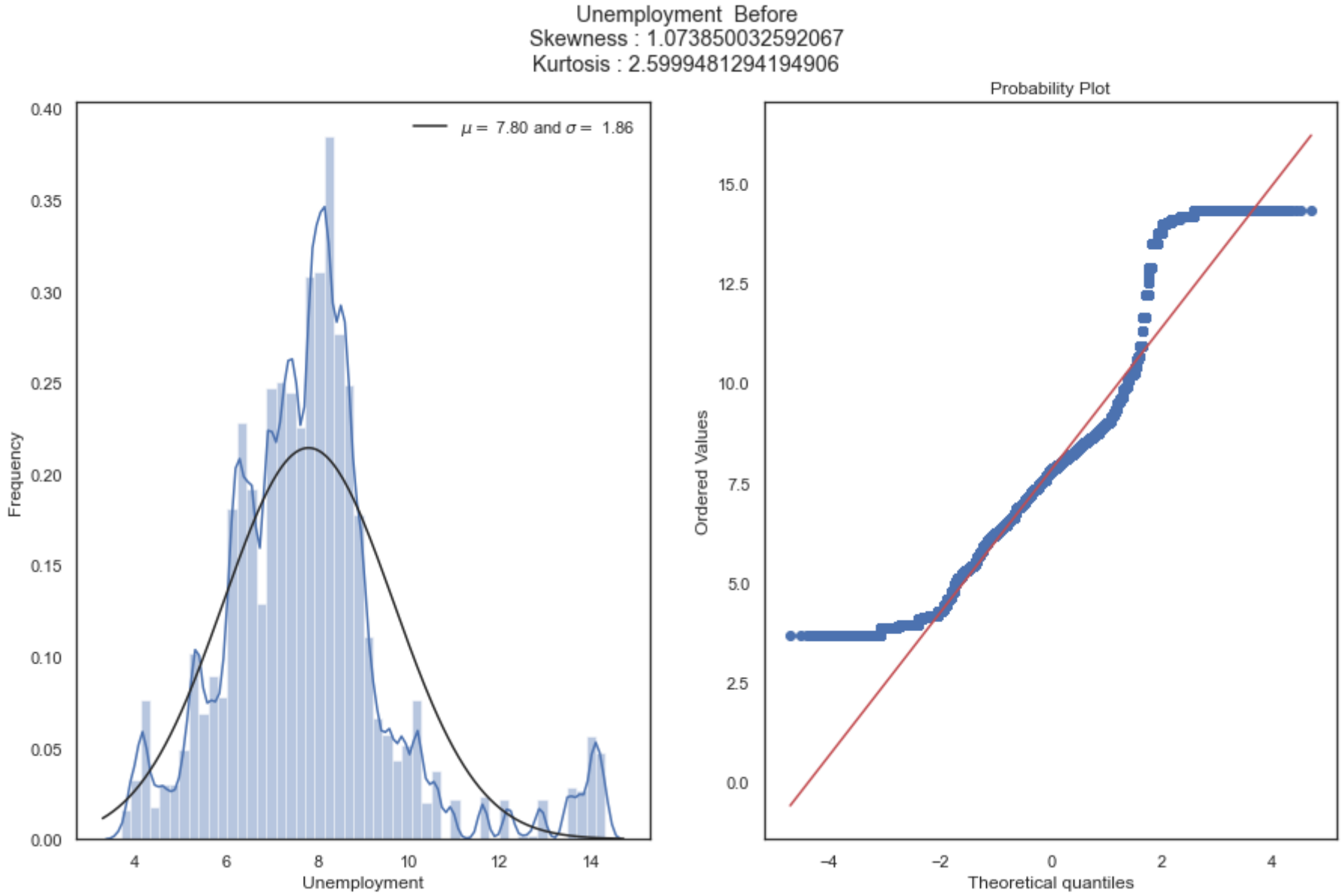


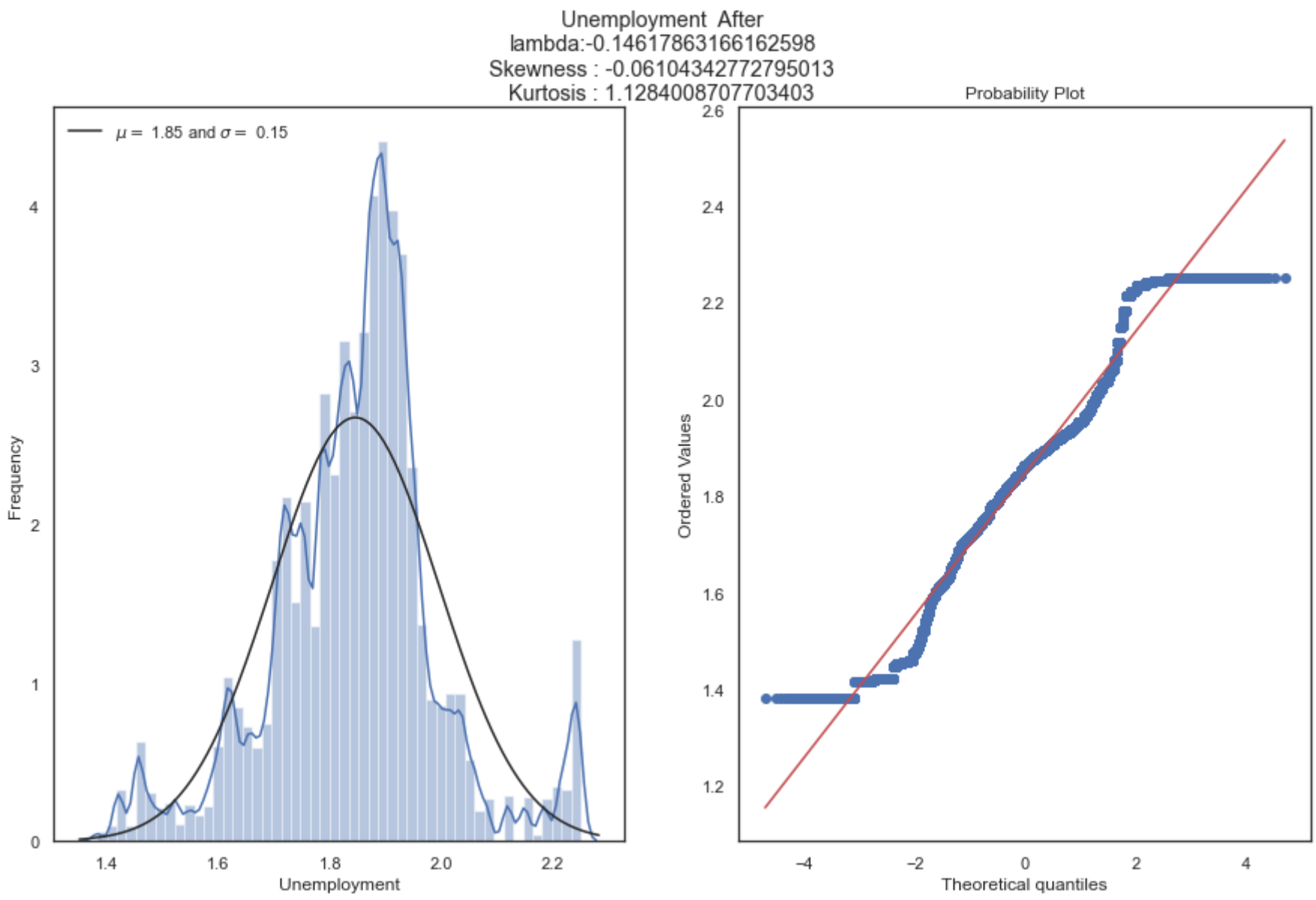


CPI After
lambda:6.014309311820474
Skewness : 0.4518661214084853
Kurtosis : -1.4102122541818392









Feature Selection

- Feature selection is to select the features that are meaningful and helpful to the model from all the features, so as to avoid the situation that all the features must be imported into the model for training.
- In fact, feature selection is the most critical and difficult step in data mining, essentially because the features used in different mining analysis tasks may be different in different scenes. There is no general rule for this selection standard. Filtering, wrapping and embedded feature selection methods are just different selection ideas proposed by various attempts. The actual situation, in fact, is to see the actual effect to know. It's not unusual for different methods to pick out different features.

Wrapper feature selection method

- The wrapper trains the evaluator on the initial feature set and obtains the importance of each feature either through the cofe_ attribute or through the feature*importances* attribute. Then, trim the least important features from the current set of features. This process is repeated recursively over the pruned set until the desired number of features are finally reached.

1. Note that the algorithm applicable to this method is not the data regression algorithm we will eventually import (i.e., not the random forest), but the professional data mining algorithm, namely the objective function. The core function of these data mining algorithms is to select the best feature subset.
2. The most typical objective function is Recursive feature Elimination (RFE). It is a greedy optimization algorithm designed to find the subset of features with the best performance. It creates the model over and over again, retaining the best features or eliminating the worst features in each iteration, and in the next iteration, it builds the next model with features that were not selected in the previous model, until all the features are exhausted. It then ranks the features according to the order in which it retains or removes them, ultimately selecting the best subset. The effect of the packaging method is the most conducive to improving the performance of the model of all feature selection methods, it can use fewer features to achieve very good results.

Parameters of Recursive Feature Elimination (RFE) : class sklearn.feature_selection.RFE (estimator, n_features_to_select=None, step=1, verbose=0)

- The parameter estimator is the instantiated estimator that needs to be filled in.

1. n_features_to_select is the number of features you want to select
2. step represents the number of features you want to remove in each iteration
3. In addition, the RFE class has two very important properties: 1) .support *: the Boolean matrix that returns whether all the features were finally selected or not, 2) .Ranking* :returns a ranking of the features by their overall importance over several iterations.

The training data set is now separated from the 'df' data set

In [110]:
```
train = df[df['Set']=='Train'].reset_index(drop=True)
train = train.drop(columns='Set')
```

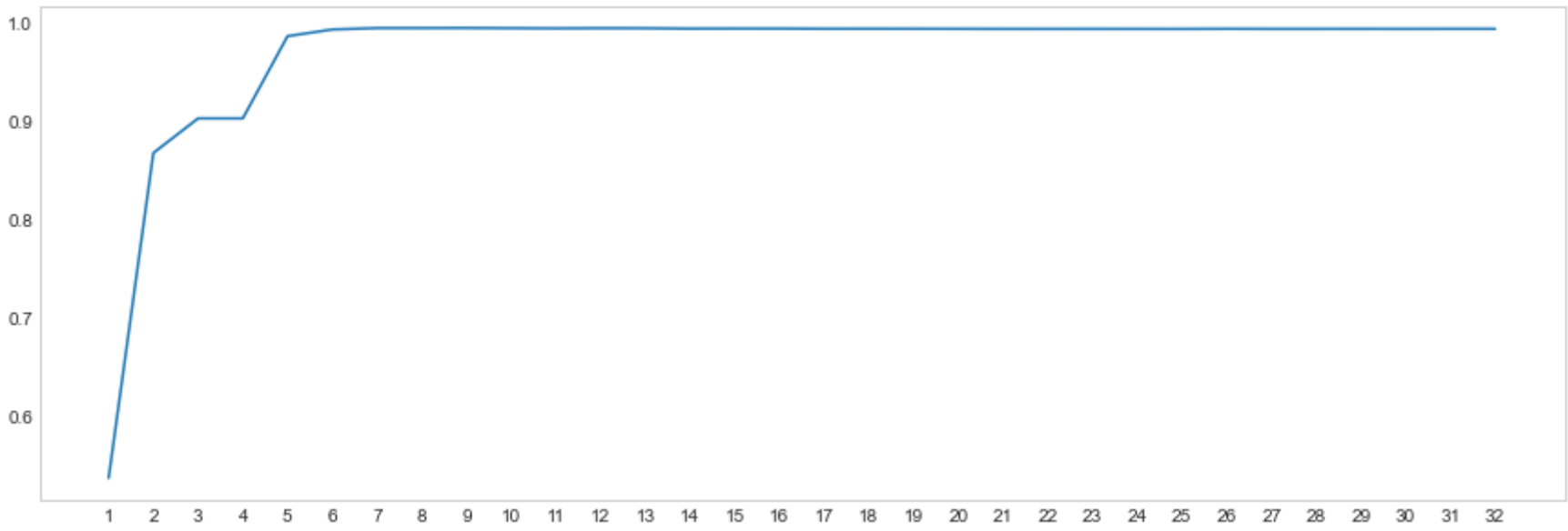
Train with the random forest model to see its preferred features

In [111]:
```
X = train.drop(columns=['Weekly_Sales','Date'])
Y = train['Weekly_Sales']
```

In [112]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 32 columns):
 #   Column                                Non-Null Count   Dtype  
---  ------                                --------------   -----  
 0   Store                                 421570 non-null  int64  
 1   Dept                                  421570 non-null  int64  
 2   Size                                  421570 non-null  int64  
 3   Temperature                           421570 non-null  float64
 4   Fuel_Price                            421570 non-null  float64
 5   MarkDown1                             421570 non-null  float64
 6   MarkDown2                             421570 non-null  float64
 7   MarkDown3                             421570 non-null  float64
 8   MarkDown4                             421570 non-null  float64
 9   MarkDown5                             421570 non-null  float64
 10  CPI                                   421570 non-null  float64
 11  Unemployment                          421570 non-null  float64
 12  markdown_sum                          421570 non-null  float64
 13  markdown_mean                         421570 non-null  float64
 14  markdown_std                          421570 non-null  float64
 15  markdown_median                       421570 non-null  float64
 16  Temperature_Category_Cold             421570 non-null  uint8  
 17  Temperature_Category_Comfortable      421570 non-null  uint8  
 18  Temperature_Category_ExtremelyCold    421570 non-null  uint8  
 19  Temperature_Category_ExtremelyHot     421570 non-null  uint8  
 20  Temperature_Category_Hot              421570 non-null  uint8  
 21  Holiday_Christmas                     421570 non-null  uint8  
 22  Holiday_Labor Day                     421570 non-null  uint8  
 23  Holiday_Super Bowl                    421570 non-null  uint8  
 24  Holiday_Thanksgiving                  421570 non-null  uint8  
 25  IsHoliday                             421570 non-null  float64
 26  Year                                  421570 non-null  int64  
 27  Month                                 421570 non-null  int64  
 28  Week                                  421570 non-null  int64  
 29  Type_A                                421570 non-null  uint8  
 30  Type_B                                421570 non-null  uint8  
 31  Type_C                                421570 non-null  uint8  
dtypes: float64(14), int64(6), uint8(12)
memory usage: 69.2 MB
```

In [209]:
```
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.feature_selection import RFE
RFR_ = RFR(n_estimators=10,n_jobs=-1,random_state=1)
scores = []
for i in range(1,33,1):
    x_wrapper = RFE(RFR_,n_features_to_select=i,step=1)
    once = x_wrapper.fit(X,Y).score(X,Y)
    scores.append(once)
plt.figure(figsize=[15,5])
plt.plot(range(1,33,1),scores)
plt.xticks(range(1,33,1))
plt.grid()
plt.show()
```



It can be seen from the learning curve that, under the current given parameters, the effect of the model tends to be stable after the number of features exceeds 6. In order to minimize the time cost of the training model, the number of features is selected to be 7.

In [210]:
```
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.feature_selection import RFE
from sklearn.model_selection import cross_val_score
RFR_ = RFR(n_estimators=10,n_jobs=-1,random_state=0)
selector = RFE(RFR_,n_features_to_select=7,step=1)
selector = selector.fit(X,Y)
```

In [557]:
```
selector.support_
```

Out[557]:
```
array([ True,  True,  True, False, False, False, False, False, False,
       False,  True, False, False,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False,  True,  True, False, False])
```

In [592]:
```
X.columns.to_list()
```

Out[592]:
```
['Store',
 'Dept',
 'Size',
 'Temperature',
 'Fuel_Price',
 'MarkDown1',
 'MarkDown2',
 'MarkDown3',
 'MarkDown4',
 'MarkDown5',
 'CPI',
 'Unemployment',
 'markdown_sum',
 'markdown_mean',
 'markdown_std',
 'markdown_median',
 'Temperature_Category_Cold',
 'Temperature_Category_Comfortable',
 'Temperature_Category_ExtremelyCold',
 'Temperature_Category_ExtremelyHot',
 'Temperature_Category_Hot',
 'Holiday_Christmas',
 'Holiday_Labor Day',
 'Holiday_Super Bowl',
 'Holiday_Thanksgiving',
 'IsHoliday',
 'Year',
 'Month',
 'Week',
 'Type_A',
 'Type_B',
 'Type_C']
```

It is necessary to include the 'IsHoliday' feature as well, as the model evaluation needs to be given five times the weight of the holiday.

In [116]:
```
predictors_weight = ['Store','Dept','Size','CPI','markdown_mean','Week','Type_A','Weekly_Sales','IsHoliday']
```

Preparing training and testing dataset

Split the train and test datasets

In [117]:
```
train = df[df['Set']=='Train'].reset_index(drop=True)
train = train[predictors_weight]
test = df[df['Set']=='Test'].reset_index(drop=True)
test = test[predictors_weight]
```

In [118]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 421570 entries, 0 to 421569
Data columns (total 9 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   Store           421570 non-null  int64  
 1   Dept            421570 non-null  int64  
 2   Size            421570 non-null  int64  
 3   CPI             421570 non-null  float64
 4   markdown_mean   421570 non-null  float64
 5   Week            421570 non-null  int64  
 6   Type_A          421570 non-null  uint8  
 7   Weekly_Sales    421570 non-null  float64
 8   IsHoliday       421570 non-null  float64
dtypes: float64(4), int64(4), uint8(1)
memory usage: 26.1 MB
```

In [119]:
```
train.shape
```

Out[119]:
```
(421570, 9)
```

In [120]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115064 entries, 0 to 115063
Data columns (total 9 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   Store          115064 non-null  int64  
 1   Dept           115064 non-null  int64  
 2   Size           115064 non-null  int64  
 3   CPI            115064 non-null  float64
 4   markdown_mean  115064 non-null  float64
 5   Week           115064 non-null  int64  
 6   Type_A         115064 non-null  uint8  
 7   Weekly_Sales   0 non-null       float64
 8   IsHoliday      115064 non-null  float64
dtypes: float64(4), int64(4), uint8(1)
memory usage: 7.1 MB
```

In [121]:
```
test.shape
```

Out[121]: (115064, 9)

Separate the labels and features of the training dataset.

Weekly_Sales is the label for the dataset.

In [122]:
```
X_train = train.drop(columns=['Weekly_Sales'])
y_train = train['Weekly_Sales']
```

Split the training dataset again

In machine learning, we usually split the training dataset into a "validation dataset" and a "training dataset" in proportion, usually using the train_test_split module in sklear.model_selection to split the data. This module will split both the feature dataset and the label dataset into their own new training and validation feature dataset and label dataset. Generally speaking, the feature dataset and the label dataset are divided into four parts.

- test_size: represents the sample proportion; If equal to 0.2, it means that 20 percent of the original training data set is taken out to generate the validation data set.
- random_state: represents the seed of random number, which is actually the number of the group of random numbers. When the experiment needs to be repeated, the same group of random Numbers can be guaranteed to be obtained. Let's say you fill in 1 every time, and you get the same random set of numbers with the same other parameters. But zero or no, it's going to be different every time.

In [123]:
```
X_train_2, X_valid, Y_train_2, Y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=14)
```

Model

Create a model evaluation function

The competition is evaluated on the weighted mean absolute error(WMAE):

$$WMAE = \frac{1}{\sum w_i} \sum_{i=1}^{n} w_i \mid y_i - \hat{y}_i \mid$$

where

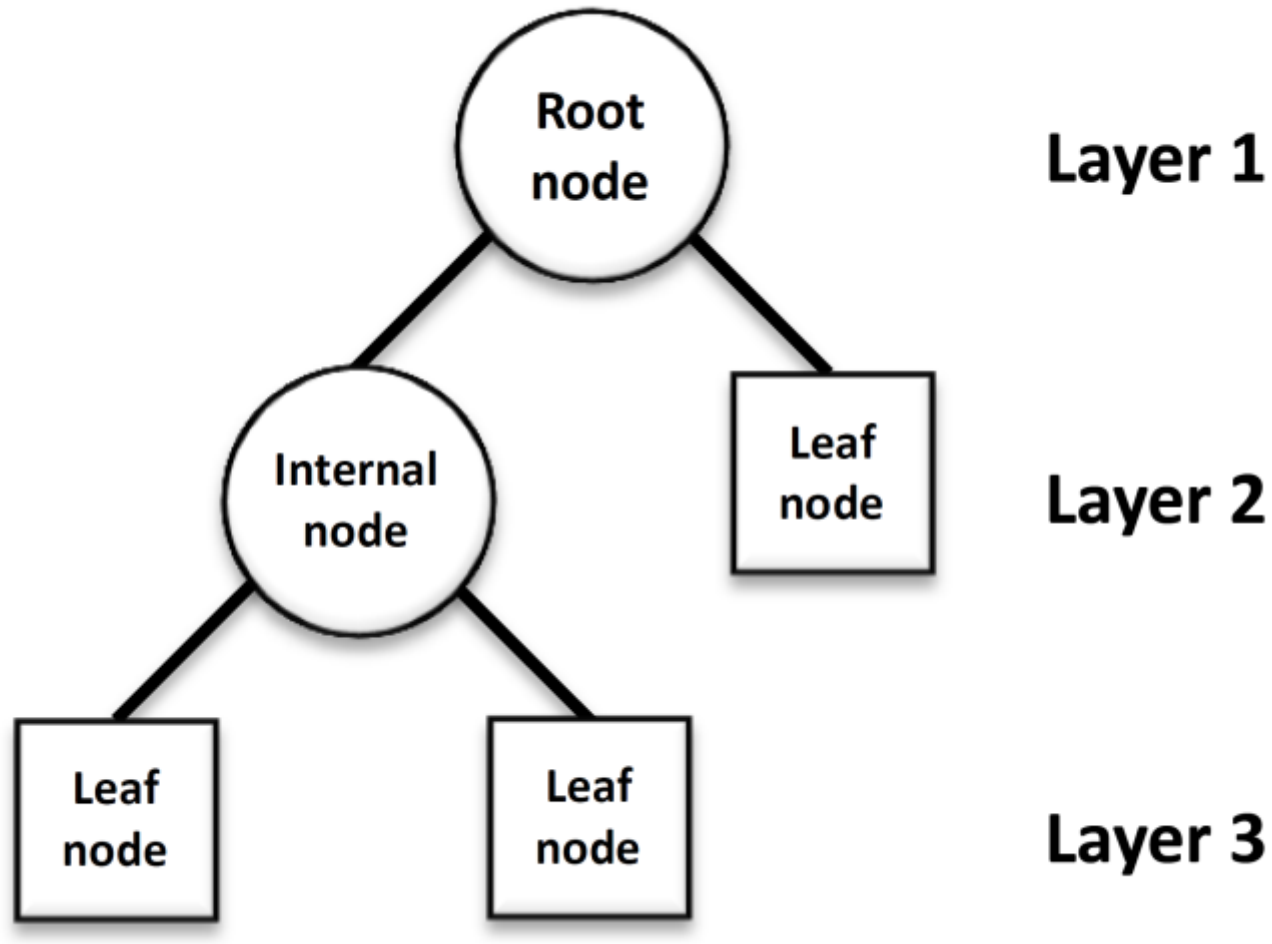
- n is the number of rows
- $\hat{y}_i$ is the predicted sales
- y_i is the actual sales
- w_i are weights. w = 5 if the week is a holiday week, 1 otherwise

In [124]:
```
ishol = X_valid.IsHoliday
def WMAE(dataset, real, predicted):
    weights = ishol.apply(lambda x: 5 if x else 1)
    return np.round(np.sum(weights*abs(real-predicted))/(np.sum(weights)), 2)
```

DecisionTreeRegressor

Definition of Desision Tree:

Decision Tree is a non-parametric supervised learning method, which can summarize Decision rules from a series of data with features and labels, and present these rules with the structure of Tree graph, so as to solve classification and regression problems. Decision tree algorithm is easy to understand, applicable to all kinds of data, and has a good performance in solving all kinds of problems. In particular, a variety of integrated algorithms with tree model as the core have been widely used in various industries and fields.



Reference: THE USE OF FACIAL MICRO-EXPRESSION STATE AND TREE-FOREST MODEL FOR PREDICTING CONCEPTUAL-CONFLICT BASED CONCEPTUAL CHANGE - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Basic-structure-of-a-decision-tree-All-decision-trees-are-built-through-recursion_fig3_295860754 (https://www.researchgate.net/figure/Basic-structure-of-a-decision-tree-All-decision-trees-are-built-through-recursion_fig3_295860754) [accessed 2 Apr, 2020]

During this decision-making process, we have been asking questions about the characteristics of the records. The place where the initial problem is called the root node, each question before the conclusion is an internal node, and each conclusion is called a leaf node.

- Root node: No input branch, only output branch. That's the initial question about the characteristics.
- Intermediate nodes: there are both input branches and output branches. There is only one input branches and many output branches. It's all about features.
- Leaf node: With input branches and no output branches, each leaf node is a label.
- Child nodes and parent nodes: Of the two contiguous nodes, the parent is closer to the root and the child is the other.

The main work of the decision tree:

Almost all model adjustment methods related to decision tree are developed around these two problems:

1. How to find the best node and the best branch from the data table?
2. High sensitivity to data noise (poor predictive stability). How to stop the growth of the decision tree and prevent overfit?

Overfit

Given a hypothesis space H and a hypothesis h belongs to H, if there are other hypotheses h'belongs to H, so that the error rate of h on the training sample is less than that of h', but h'is less than that of h on the whole instance distribution, then it is assumed that h overfits the training data. -- Tom M.Mitchell, Machine Learning

Build decision tree

In principle, all features on any dataset can be branched, and any nodes on the features can be combined freely. Therefore, a large number of decision trees can be developed on a dataset, and the number of them can reach exponential level. There is always one tree that regresses better than the others, and that tree is called the global optimal tree.

- Global optimal: the model with the best regression effect as a whole
- Locally optimal: each time it branches, it branches towards a better regression effect, but it is not possible to determine whether the resulting tree is globally optimal

Greedy strategy

It is impossible to find the one with the best classification among so many decision trees at once, and it would be too computation-intensive and inefficient to filter by permutation and combination, so we would not do it. In contrast, machine learning researchers have developed some effective algorithms that can construct suboptimal decision trees with certain accuracy in a reasonable amount of time. These algorithms basically execute the "greedy strategy", which is to use local optimality to achieve what we believe to be the closest to global optimality, which is what all tree models do.

Regression decision tree

Regression decision tree mainly refers to CART(classification and regression tree) algorithm. The value of internal node features is "yes" and "no", and is a binary tree structure.

The so-called regression is to determine the corresponding output value according to the eigenvector. A regression tree is a partition of a feature space into units, each of which has a specific output. Since each node is a "yes" and "no" judgment, the boundary is parallel to the coordinate axis. For test data, we simply attribute it to a unit and get the corresponding output value.The process of partition is also the process of building a tree. Each partition, then determine the corresponding output of the partition unit, also one more node. When you break it up according to the stop condition, the output of each cell is determined, the leaf nodes.

- Split point selection: If the target variable is continuous, then for node M, R_m represents a region with N_m observations. The general principle for determining the next segmentation is to minimize the Mean Squared Error, that is, to use the Mean value of the end point and minimize the L2 Error. Or minimize the Mean Absolute Error, i.e., minimize the L1 Error by using the median value of the end point.
- Output value: intra-cell mean

Mean square error (mse):

$$H(X_m) = \frac{1}{N_m} \sum_{i \in \frac{1}{Nm}} (y_i - c_m)^2$$

$$c_m = \frac{1}{N_m} \sum_{i \in N_m} y_i$$

X_m is the training data at the node m.

Mean absolute error (mae):

$$H(X_m) = \frac{1}{N_m} \sum_{i \in \frac{1}{Nm}} |y_i - \hat{y}_m|$$

$$\hat{y}_m = \frac{1}{N_m} \sum_{i \in N_m} y_i$$

X_m is the training data at the node m.

Algorithm described

1. Input: Training dataset D
2. Output: Regression tree $f(x)$
3. In the input space where the training data set is located, each region is recursively divided into two sub-regions and the output value on each sub-region is determined to build a binary decision tree:

1) The optimal segmentation variables j and the segmentation point s are selected and solved:

$$\min_{j,s}[\min_{c1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2]$$

Map the variable j, scan the segmentation point s for the fixed segmentation variable j, and select the value pair (j, s) that makes the above equation reach the minimum value.

2) Divide the region with the selected value pair (j, s) and determine the corresponding output value:

$$\hat{c_m} = \frac{1}{N_m} \sum_{x_i \in R_m(j,s)} y_i, \ (x \in R_m, m = 1, 2)$$

Among them,

$$R_1(j, s) = \{x \mid x^{(j)} \leq s\}, \ R_2(j, s) = \{x \mid x^{(j)} > s\}$$

3) Continue calling steps (1),(2) on the two subregions until the stop condition is met;

4) Divide the input space into M areas $R_1, R_2 \ldots, R_M$, to generate the decision tree:

$$f(x) = \sum_{m=1}^{M} \hat{c_m} I(x \in R_m)$$

Where I is the indicator function,

$$I = \begin{cases} 1 & \text{if}(x \in R_m) \\ 0 & \text{if}(x \notin R_m) \end{cases}$$

Advantages of decision tree

- Easy to understand and explain, because we can draw, many other algorithms can not do this. It's a white-box model, and the results can be easily explained. If a given situation can be observed in the model, the result can be easily interpreted through Boolean logic. Conversely, in a black-box model (for example, in an artificial neural network), the results may be more difficult to interpret.
- Little data preparation is required. Many other algorithms typically require data normalization, creating virtual variables and removing null values, among other things.
- Compared with other algorithms, the time cost of using trees is very low.
- Able to process both Numbers and classified data, which can be used for both regression and classification. Other techniques are often specialized for analyzing data sets that have only one variable type.

- Able to handle multiple output problems, that is, problems with multiple labels, and note the difference between problems with multiple label categories in a single label.
- Statistical tests can be used to validate the model, which allows us to consider the reliability of the model.

Disadvantages of decision tree

- Decision tree learners may create overly complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required for leaf nodes, or setting the maximum depth of the tree are necessary to avoid this problem, but are time-consuming.
- The decision tree may be unstable, and small changes in the data may lead to the generation of completely different trees. This problem needs to be solved by integrating algorithms.
- The learning of decision tree is based on the greedy algorithm, which tries to achieve the overall optimal by optimizing the local optimal (the optimal of each node), but this method cannot guarantee the return of the global optimal decision tree. This problem can also be solved by the integration algorithm, in the random forest, features and samples will be randomly sampled in the branching process.
- The decision tree module in Sklearn does not support processing of missing values.

Import the decision tree module

In [75]:
```
from sklearn.tree import DecisionTreeRegressor
```

Instantiate the decision tree model

In [76]:
```
dt = DecisionTreeRegressor()
```

Train the decision tree model

In [77]:
```
dt.fit(X_train_2,Y_train_2)
```

Out[77]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

WMAE of the trained model

In [80]:
```
predD = dt.predict(X_valid)
print(WMAE(X_valid,Y_valid,predD))
```

```
2198.61
```

Decision coefficient(R^2) of the Model

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y})^2}{\sum (y_i - \bar{y})^2}$$

Where, the molecular part represents the sum of the square error of the real value and the predicted value, similar to MSE; The denominator is the sum of the squares of the true value and the mean, which is kind of like the variance.

According to the value of r-squared, the model is judged to be good or bad. Its value range is [0,1] : If the result is 0, the model fitting effect is poor; If the result is 1, the model is error free.

Normally, the larger R-squared is, the better the model fitting effect is. R-squared tells you roughly how accurate it is, because as the number of samples increases, R-square increases, and you can't really quantify how accurate it is, you can only approximate it.

In [81]:
```
dt.score(X_valid,Y_valid)
```

Out[81]:
```
0.9577293605846362
```

> In fact, with the help of the decision tree model, the prediction accuracy has reached more than 95.77%. However, based on the instability of the decision tree, it is more safe to make the prediction through the emsemble algorithm. I'm using the random forest model here.

RandomForestRegressor

Ensemble Learning

Ensemble Learning is a very popular machine learning algorithm. It is not a single machine learning algorithm itself, but integrates the modeling results of all models by building multiple models on the data. This algorithm considers the modeling results of multiple evaluators, and then summarizes the results to obtain a better regression or classification performance than a single model.

- Ensemble Estimator: The ensemble Estimator is composed of several models.
- Base Estimator: Each model of the ensemble estimator is called a Base Estimator.

In general, there are three emsemble algorithms: Bagging, Boosting, and Stacking.

Bagging emsemble algorithms

The representative model of bagging emsemble algorithms is the random forest.

The idea of the Bagging emsemble algorithms: Suppose a dataset L is sampled n times through Boostrap(self-sampling), then n sample datasets are generated, denoted as n train sets.

- Step 1: We train the n trains, so that n basis evaluators, or n decision trees, are generated
- Step 2: Use these N base evaluators to predict the test set, so that n results will be obtained.
- Step 3: Sum and average all the results: Sum the results of n predictions and calculate the average value as the population forecast.

Some new problems arise from careful consideration. If a data set has a strong predictor and some moderate predictor, then it can be expected that most (or even all) trees will use the strongest predictor for the top split point, which will cause all bagging trees to look similar. Averaging many highly correlated quantities does not bring about the same degree of variance reduction as averaging unrelated quantities. In this case, bagging does not result in a significant reduction in variance compared to a single tree. This problem is a fatal one in bagging. So let's look at random Forests Methods.

Random Forest

Random forest is an improvement of bagging method, and it also needs to establish a series of decision trees for the self-help sampling training set, which is similar to decision trees. However, random forests in the build tree, not bagging method, bagging method when done is to take into account all the prediction variables, and the random forest is considered points of each division, were randomly selected from all the predictor variable p m a prediction variables, prediction variables used in the split point can only select one of the m variable. At each split point, re-sampling was performed to select m predictive variables, usually $m \approx \sqrt{p}$. For each split point, this algorithm excluded most of the available predictive variables, which sounded crazy, but the principle was clever.

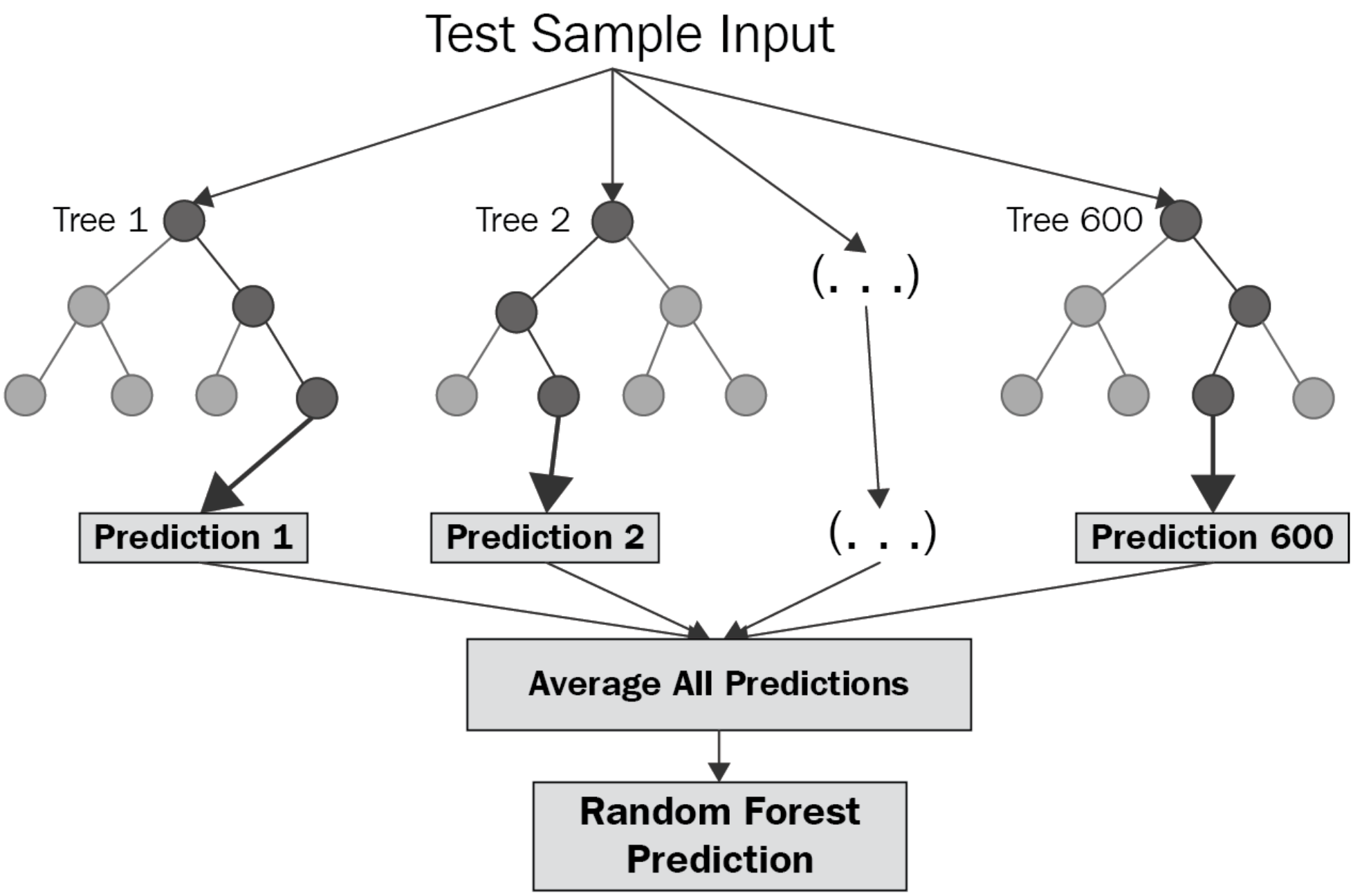


Image Source (https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f)

In fact, when m is equal to p in the random forest, the random forest and the bagging method are the same. The random forest considers a subset of each split point relatively less than the bagging method. The average value of the resulting tree has a smaller variance, so the tree is relatively reliable.

Random Forest Regressor class sklearn.ensemble.RandomForestRegressor(n_estimators=100, *, criterion='mse', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None)

- criterion: {“mse”, “mae”}, default=”mse”

> The function to measure the quality of a split. Supported criteria are “mse” for the mean squared error, which is equal to variance reduction as feature selection criterion, and “mae” for the mean absolute error.

- n_estimators: int, default=100

> The number of trees in the forest.

- max_depth: int, default=None

> The maximum depth of the tree, branches above the maximum depth will be cut off. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

- min_samples_split: int or float, default=2

> A node must contain at least min_samples_split training samples before it is allowed to branch, otherwise branching will not occur.

- min_samples_leaf: int or float, default=1

> Each child node of a node after branching must contain at least min_samples_split training samples, otherwise branching will not occur.

- max_features: {“auto”, “sqrt”, “log2”}, int or float, default=”auto”

> The number of features to consider when looking for the best split.

- random_state: int or RandomState, default=None

> The model construction is a random process, after fixing random_state, the model is the same every time. For processes that are essentially random, it is necessary to control the random state so that the same results can be repeated.

- n_jobs: int, default=None

> To set the CPU running state, n_jobs=-1 is to use the entire CPU.

A random forest model with default parameters

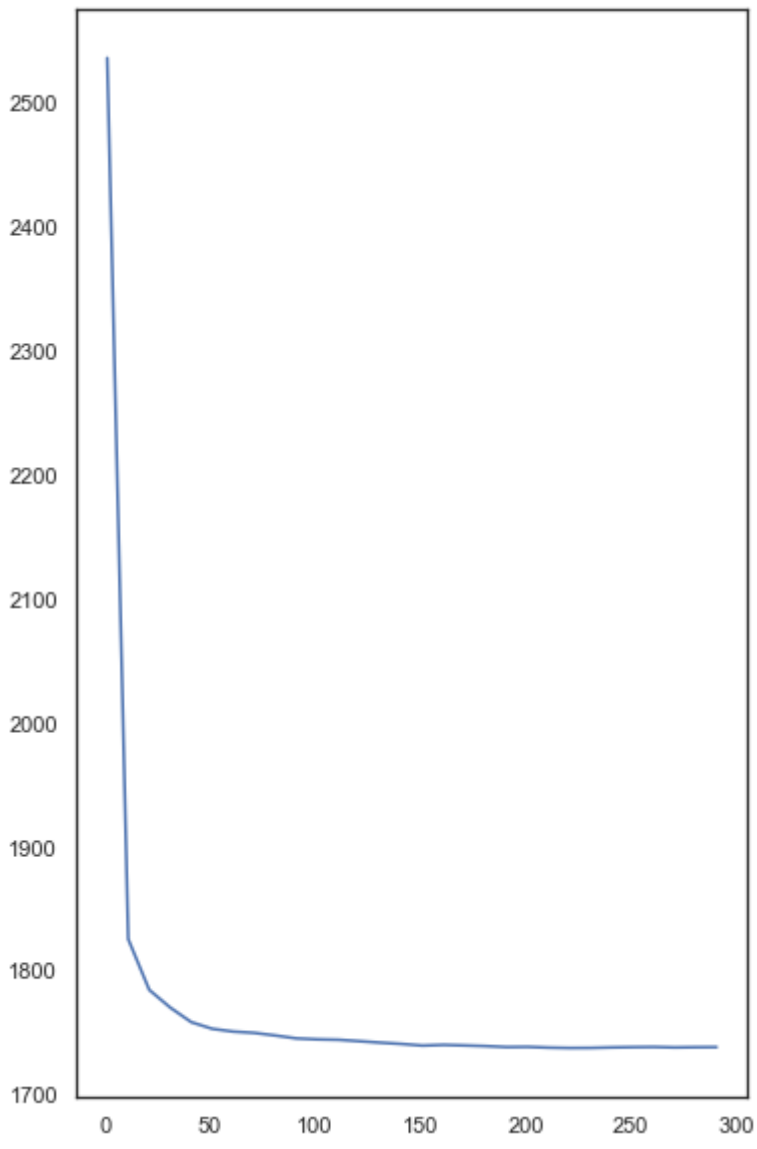
In [83]:
```
from sklearn.ensemble import RandomForestRegressor
wmaes = []
scores = []
for i in range(1,6):
    rfr = RandomForestRegressor(n_jobs=-1)
    rfr.fit(X_train_2,Y_train_2)
    predR = rfr.predict(X_valid)
    wmae = WMAE(X_valid,Y_valid,predR)
    wmaes.append(wmae)
    score = rfr.score(X_valid,Y_valid)
    scores.append(score)
print(np.mean(wmaes))
print(np.mean(scores))
```

```
1838.7279999999998
0.9706944750824779
```

Adjust the number of trees in the forest

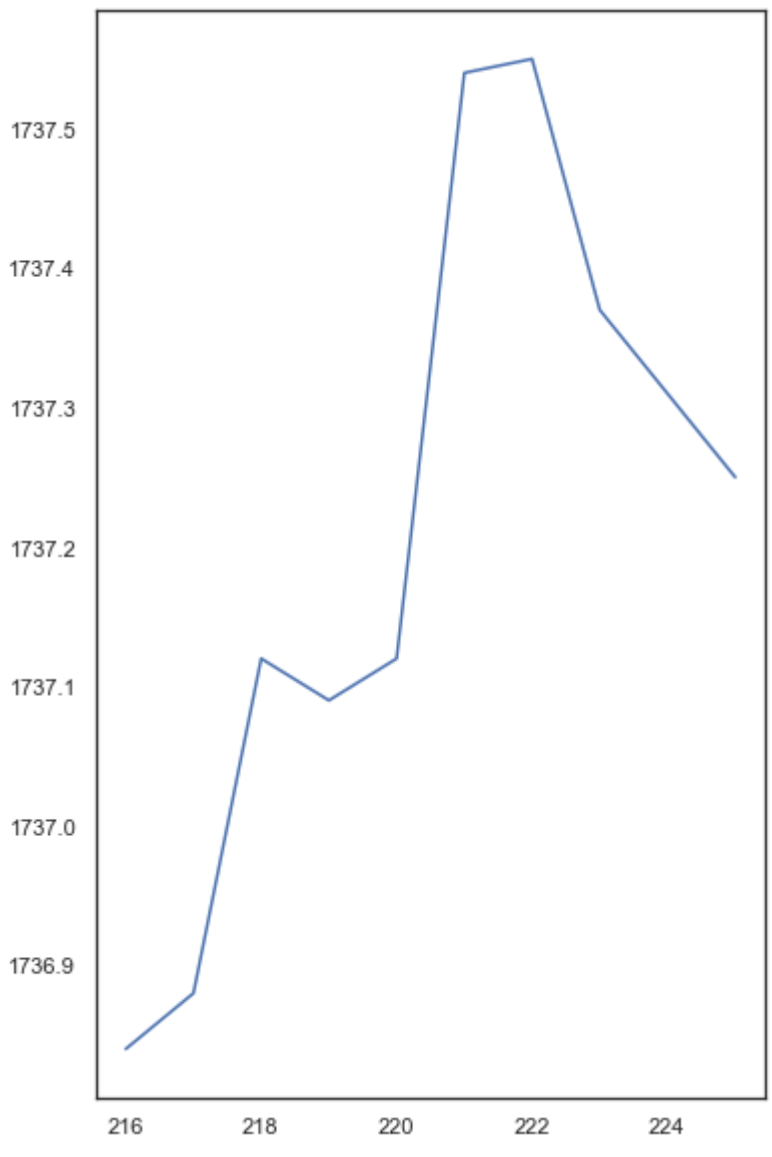
In [517]:
```
scorel = []
for i in range(1,301,10):
    rfr = RandomForestRegressor(n_estimators=i,
                                n_jobs=-1,
                                random_state=90)
    rfr.fit(X_train_2,Y_train_2)
    predR = rfr.predict(X_valid)
    score = WMAE(X_valid,Y_valid,predR)
    scorel.append(score)

print(min(scorel),(scorel.index(min(scorel))*10)+1)

plt.figure(figsize=[6,10])
plt.plot(range(1,301,10),scorel)
plt.show()
```

```
1737.54 221
```



In [518]:
```
scorel = []
for i in range(216,226,1):
    rfr = RandomForestRegressor(n_estimators=i,
                                n_jobs=-1,
                                random_state=90)
    rfr.fit(X_train_2,Y_train_2)
    predR = rfr.predict(X_valid)
    score = WMAE(X_valid,Y_valid,predR)
    scorel.append(score)

print(min(scorel),([*range(216,226,1)][scorel.index(min(scorel))]))

plt.figure(figsize=[6,10])
plt.plot(range(216,226,1),scorel)
plt.show()
```

```
1736.84 216
```

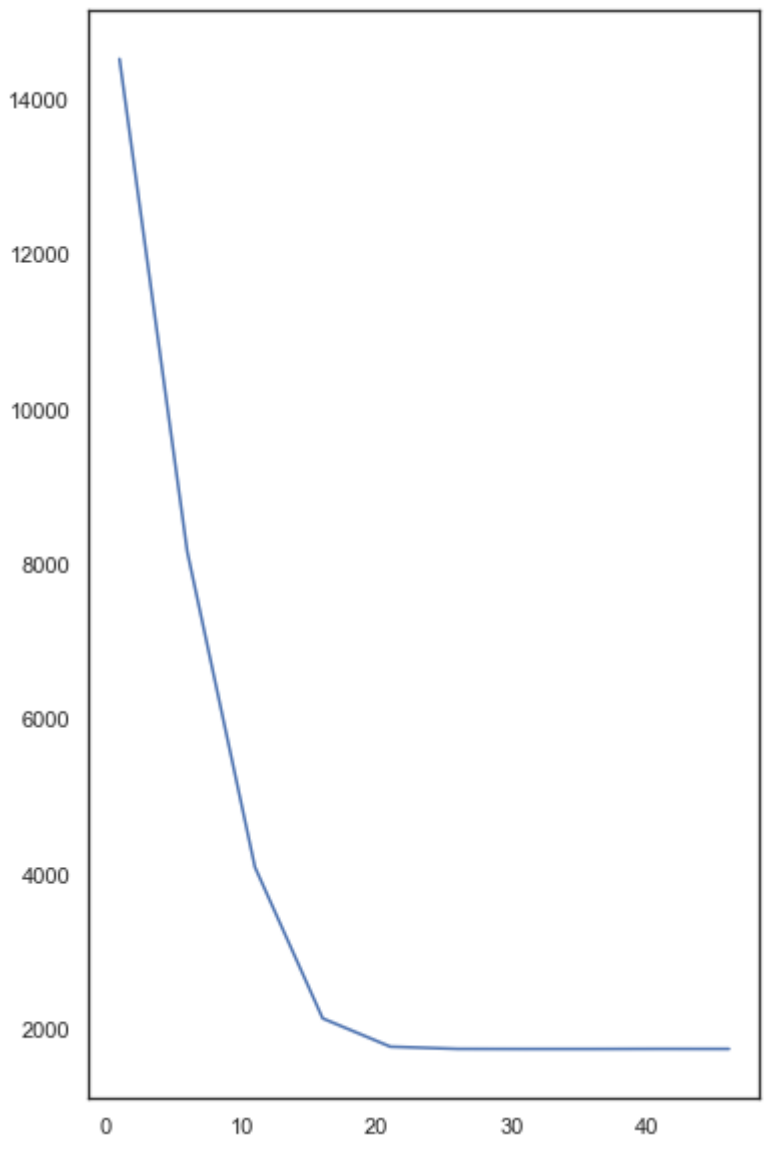


WMAE is the lowest when N_estimators = 282.

Adjust the value of the parameter max_depth

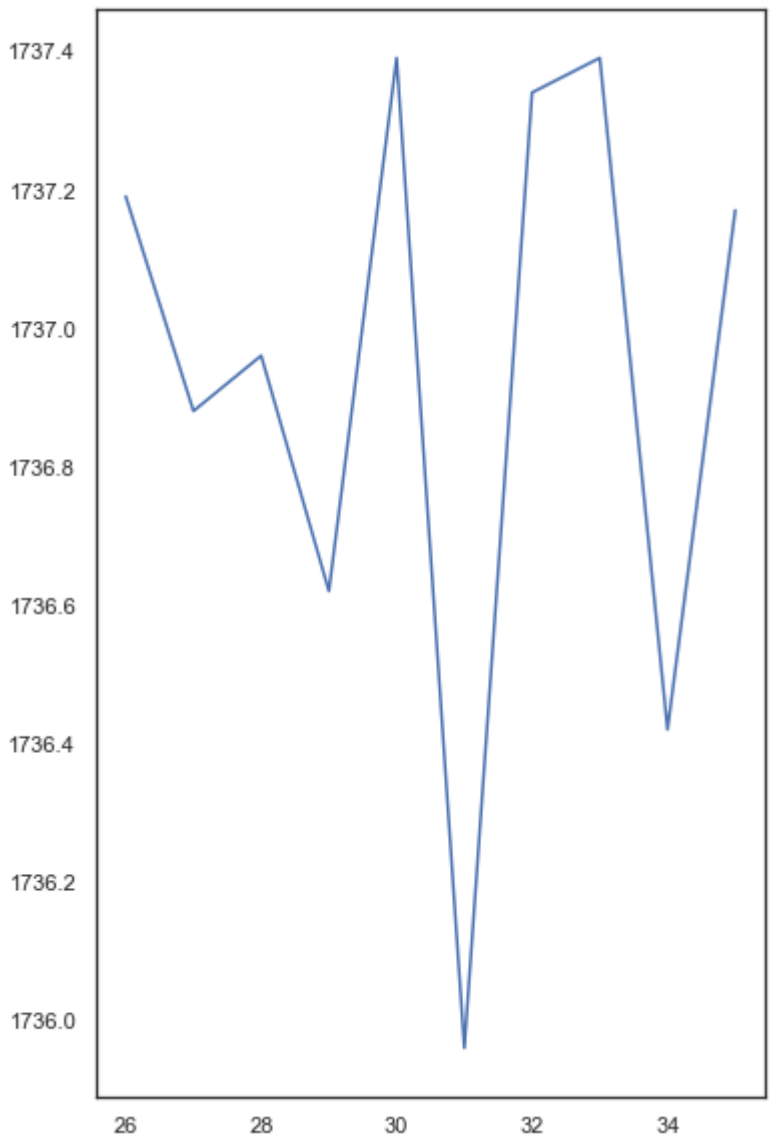
In [519]:
```
scorel = []
for i in range(1,51,5):
    rfr = RandomForestRegressor(n_estimators=216,
                                max_depth = i,
                                n_jobs=-1,
                                random_state=90)
    rfr.fit(X_train_2,Y_train_2)
    predR = rfr.predict(X_valid)
    score = WMAE(X_valid,Y_valid,predR)
    scorel.append(score)

print(min(scorel),([*range(1,51,5)][scorel.index(min(scorel))]))

plt.figure(figsize=[6,10])
plt.plot(range(1,51,5),scorel)
plt.show()
```

```
1735.96 31
```



In [520]:
```
scorel = []
for i in range(26,36,1):
    rfr = RandomForestRegressor(n_estimators=216,
                                max_depth = i,
                                n_jobs=-1,
                                random_state=90)
    rfr.fit(X_train_2,Y_train_2)
    predR = rfr.predict(X_valid)
    score = WMAE(X_valid,Y_valid,predR)
    scorel.append(score)

print(min(scorel),([*range(26,36,1)][scorel.index(min(scorel))]))

plt.figure(figsize=[6,10])
plt.plot(range(26,36,1),scorel)
plt.show()
```

```
1735.96 31
```

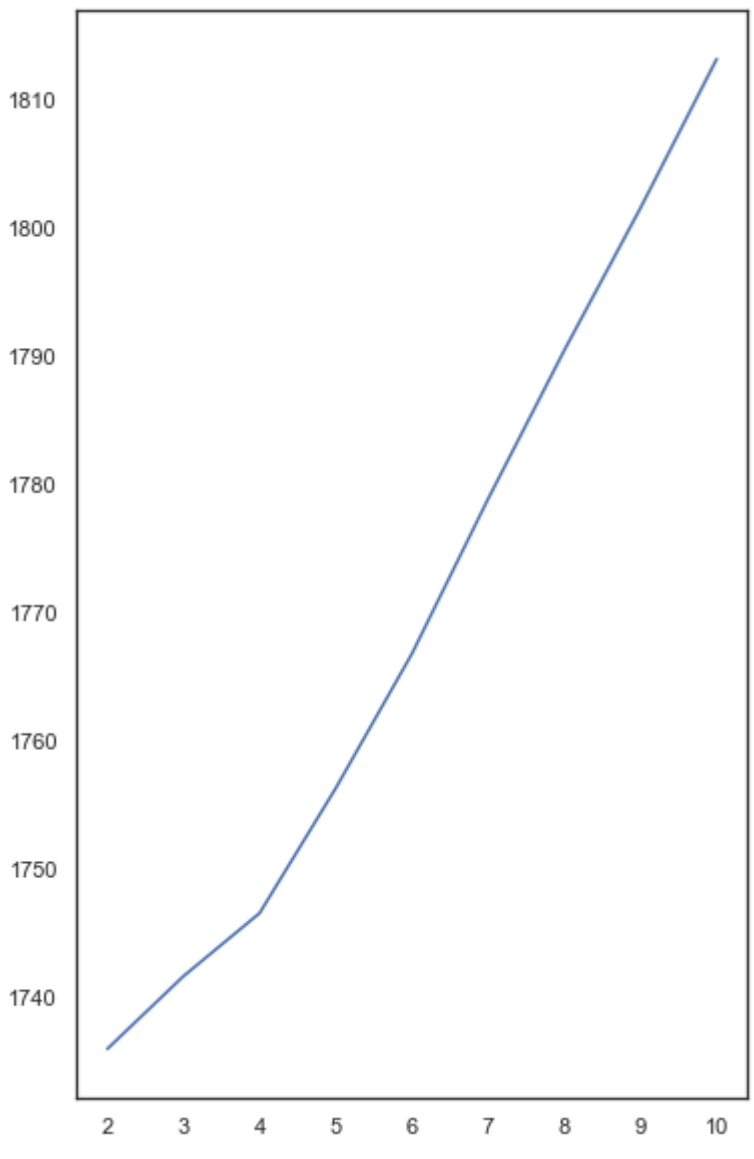


WMAE is the lowest when max_depth = 31.

Adjust the value of the parameter min_samples_split

In [521]:
```
scorel = []
for i in range(2,11):
    rfr = RandomForestRegressor(n_estimators=216,
                                max_depth = 31,
                                min_samples_split = i,
                                n_jobs=-1,
                                random_state=90)
    rfr.fit(X_train_2,Y_train_2)
    predR = rfr.predict(X_valid)
    score = WMAE(X_valid,Y_valid,predR)
    scorel.append(score)

print(min(scorel),([*range(2,11)][scorel.index(min(scorel))]))

plt.figure(figsize=[6,10])
plt.plot(range(2,11),scorel)
plt.show()
```

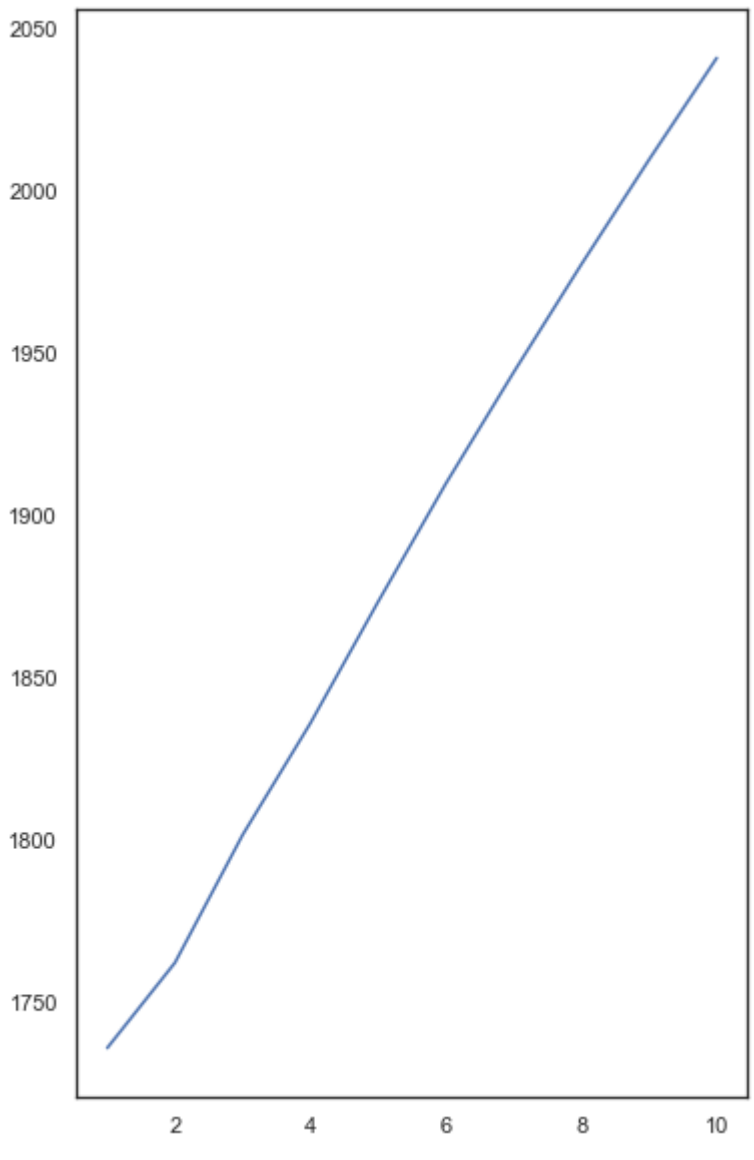
```
1735.96 2
```



Adjust the value of the parameter min_samples_leaf

In [523]:
```
scorel = []
for i in range(1,11):
    rfr = RandomForestRegressor(n_estimators=216,
                                max_depth = 31,
                                min_samples_split = 2,
                                min_samples_leaf = i,
                                n_jobs=-1,
                                random_state=90)
    rfr.fit(X_train_2,Y_train_2)
    predR = rfr.predict(X_valid)
    score = WMAE(X_valid,Y_valid,predR)
    scorel.append(score)

print(min(scorel),([*range(1,11)][scorel.index(min(scorel))]))

plt.figure(figsize=[6,10])
plt.plot(range(1,11),scorel)
plt.show()
```

```
1735.96 1
```



Final model

In [62]:
```
rfr = RandomForestRegressor(n_estimators=216,
                            max_depth = 31,
                            min_samples_split = 2,
                            min_samples_leaf = 1,
                            random_state=90,
                            n_jobs=-1)
rfr.fit(X_train,y_train)
```

Out[62]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=31,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=216, n_jobs=-1,
                      oob_score=False, random_state=90, verbose=0,
                      warm_start=False)
```

WMAE and R-squared for the final model

In [593]:
```
from sklearn.ensemble import RandomForestRegressor
wmaes = []
scores = []
for i in range(1,6):
    rfr = RandomForestRegressor(n_estimators=216,
                                max_depth = 31,
                                min_samples_split = 2,
                                min_samples_leaf = 1,
                                random_state=90,
                                n_jobs=-1)
    rfr.fit(X_train_2,Y_train_2)
    predR = rfr.predict(X_valid)
    wmae = WMAE(X_valid,Y_valid,predR)
    wmaes.append(wmae)
    score = rfr.score(X_valid,Y_valid)
    scores.append(score)
print(np.mean(wmaes))
print(np.mean(scores))
```

```
1735.9599999999998
0.9741324879101226
```

- **WMAE:** 1735.96
- **R-squared:** 0.9741

Submission

In [145]:
```
test_feature = test.drop(columns='Weekly_Sales')
```

In [146]:
```
predict = rfr.predict(test_feature)
```

In [147]:
```
Test = df[df['Set']=='Test'].reset_index(drop=True)
```

In [149]:
```
submission = test[['Store', 'Dept']]
submission['Date'] = Test['Date']
submission['Weekly_Sales'] = predict
```

In [153]:
```
submission.head()
```

Out[153]:

	Store	Dept	Date	Weekly_Sales
0	1	1	2012-11-02	65450.794769
1	1	1	2012-11-09	49536.006528
2	1	1	2012-11-16	49724.724028
3	1	1	2012-11-23	50508.688981
4	1	1	2012-11-30	55773.666019

In [152]:
```
submission.to_csv('submission.csv',index=True)
```